An Interactive Visual Query Environment for Exploring Data

Mark Derthick, John Kolojejchick, Steven F. Roth

Carnegie Mellon University
Robotics Institute, School of Computer Science
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
E-mail: {mad+, jake+, roth+}@cs.cmu.edu

ABSTRACT

Direct manipulation of visualizations is a powerful technique for performing exploratory data operations such as navigation, aggregation, and filtering. Its immediacy facilitates rapid, incremental, and reversible forays into the data. However it does not provide for reuse or modification of exploration sessions. This paper describes a visual query language, VQE, that adds these capabilities to a direct manipulation exploration environment called Visage. Queries and visualizations are dynamically linked: operations on either one immediately update the other, in contrast to the feedforward sequence of database query followed by visualization of results common in traditional systems.

These features are supported by the architectural concept of *threads*, which represent a sequence of navigation steps on particular objects. Because they are tied to particular data objects, they can be directly manipulated. Because they represent operations, they can be generalized into queries. We expect this technique to apply to direct manipulation interfaces to any object-oriented system that represents both objects and the relationships among them.

NOTE: Color versions of the figures are at, e.g., http://www.cs.cmu.edu/~sage/UIST97/figure1.gif

1. THE MULTIPLE OBJECT PROBLEM

We are concerned with data visualization systems that produce "business graphics" consisting of charts, maps, network diagrams, as well as more sophisticated and special purpose graphics showing abstract data. It is often natural to provide the user with an object-oriented data model to accompany these visualizations. For instance in Figure 1, each point on the map represents a single house. Each house has attributes like neighborhood, latitude, and number_of_rooms.

Many data exploration tasks involve the relationship among multiple objects. For instance, the database schema shown in Figure 2(top) has separate objects for houses, schools, companies, people, and sales. The schema also includes the attributes for each object type, and the relationships among objects. For example, companies have a name, total_sales, and an incorporation_date. They can be the builders of houses. Persons can fill any of four roles in a sale: seller_agent, buyer_agent, buyer, or seller.

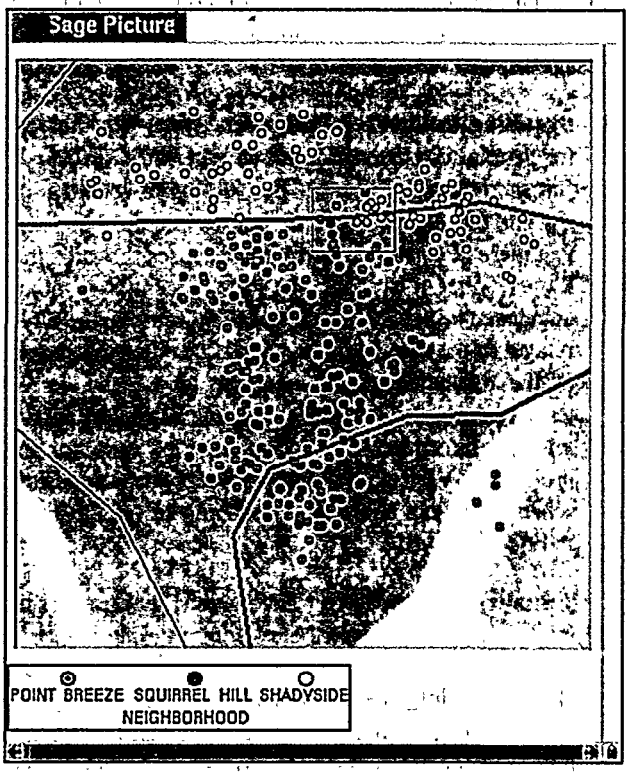

Figure 1: All houses, color coded by neighborhood

UIST 97 Banff, Alberta, Canada

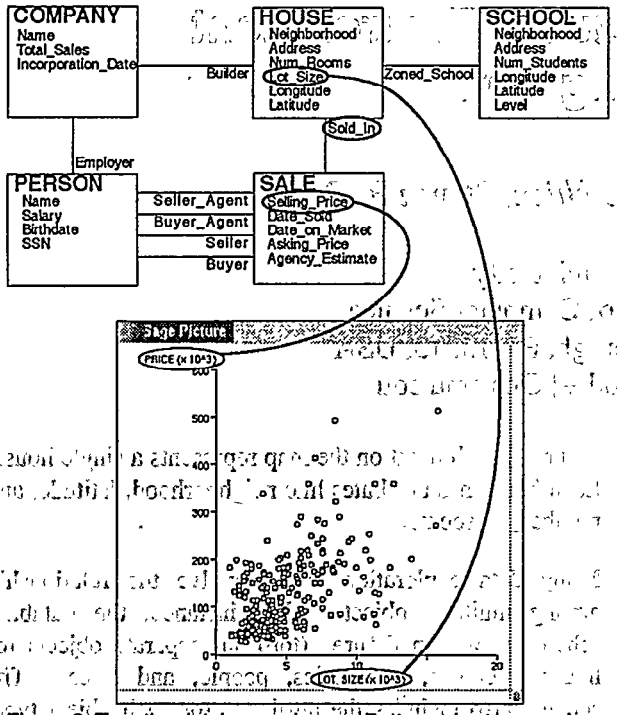

Figure 2: Top: the database schema. Bottom: chart comparing attributes from two different object types. All <house, sale> pairs related by the sold_in relationship generate a point on the chart.

Databases containing multiple objects like these increase the complexity of information exploration processes. In particular, there are three areas of complexity that we have identified and attempted to address in the Visage/VQE environment.

1. **Browsing to locate information.** First, it is difficult for people to locate information when it is distributed across multiple objects. For example, locating the current owners of a set of houses requires knowing that there is no simple attribute of houses that tells this. Rather, one must find the most recent sale object associated with each house and then retrieve the buyer related to the sale. In other words, users must be aware of the database schema in order to use the correct terms for objects, relationships, and attributes to find relevant information. We've provided help for this problem in VQE with a graphical representation of the schema that can be directed manpulated (as illustrated in Figure 2, top).

2. **Visualization.** Creating visualizations that present the relationships among attributes from multiple objects is also complex. For example, Figure 2 (bottom) is a chart of the lot_size of houses versus the selling_price of the corresponding sales of those houses. A single point encodes attributes of two objects: a house and sale. No previous interactive visualization system directly supports the construction of visualizations that are derived from multiple objects. Typically, a relational database query must be constructed using a separate interface that joins multiple tables into one containing attributes from both houses and sales. The selling_price/lot_size chart can then be generated by mapping graphical points to records in this new table. Once the new table is constructed, it is not obvious how to coordinate visualizations of it with those of related tables from which it was derived (e.g. to enable coordinated brushing of points representing houses on a map with points on a chart representing the join of houses and sales as in Figures 1 and 2). We discuss this problem in detail in Section 1.1.

3. **Expressing Queries.** These databases require users to retrieve information based on the attributes of multiple objects and/or the relationships among objects. For example, one might need to find the sales of a given house, or to find all the houses with big lots that sold for a low price. Furthermore, it is useful for queries to be either *extensional* or *intentional*, to provide different means by which users can express their requests for objects to be retrieved.

   a) **Extensional Expressions.** Often, it is easier to indicate sets of objects via direct manipulation than to create an expression defining the set. The bounding box in Figure 1 selects some houses near the border between Squirrel Hill and Point Breeze. This is an extensional operation because the set of points is selected without conveying the underlying intent of the set. Consequently, if the border were later changed and the neighborhood attribute of the affected houses were updated, no mechanism could infer the change in the selected set. In general, extensional queries can't be reused on different data.

   b) **Intentional Expressions.** An analogous intentional query would, for instance, select "all houses within 100 yards of a house with a different value for neighborhood attribute." Intentional queries have a declarative representation distinct from what they evaluate to on the current data, and so can be reused.

Our approach to supporting these processes for multi-object databases is to combine VQE, our visual query environment, with Visage, our direct manipulation data exploration environment. In support of the points above:

- VQE includes a schema browser similar to Figure 2 (top), as well as providing a graphical representation of the query which more easily allows users to find the attributes they need.
- Visage has been integrated with the SAGE system [10,11] for automated design of visualizations that integrate many attributes. SAGE is a visualization server to Visage, which renders SAGE graphic

designs so that they are subject to all direct manipulation operations.

- Visage provides direct manipulation operations that allow users to make extensional queries, while VQE allows the creation of intentional queries. Furthermore, the integration of VQE with SAGE enables the visualization of attributes derived from multiple data objects.

In the following 2 sections we describe Visage and VQE in greater detail.

1.1 Visage

Visage is an *information centric* [10,11] user interface environment for data exploration and for creating interfaces to data-intensive applications. Data objects are represented as first class interface objects that can be manipulated using a common set of basic operations, such as drill-down and roll-up, drag-and-drop, copy, and dynamic scaling. These operations are universally applicable across the environment, whether graphical objects appear in a hierarchical table, a map, a slide show, a query, or other application user interface. Furthermore, graphical objects can be dragged across application UI boundaries. Integrating the visualization system directly with an underlying database, rather than just deriving visualizations from otherwise isolated tables, is key to coordinating visualization applications with the other components of an exploratory data analysis environment.

Visage includes ubiquitous extensional query operations. A user can *navigate* from the visual representation of any database object to other related objects. For instance, from a graphical object representing a real estate sales agent, one can navigate to all the houses listed by that agent. It is also possible to *aggregate* database objects into a new object, which will have attributes derived from its elements. For instance, we could aggregate the houses listed by John and look up their average size or *recompose* this set into sub-aggregates based on neighborhood or number of rooms (i.e. all Shadyside houses listed by John with 8 rooms).

We can *brush* [3] graphical objects in a visualization to change their color, and have graphical objects in all visualizations representing the same data object also be colored. This kind of coordination enables the user to see correlations among more variables than can be encoded in a single visualization.

We can *filter* objects based on any numerical attribute with a Dynamic Query slider [1]. Graphical objects representing a data object whose attribute value falls outside the range selected by the slider are made invisible.

The three direct manipulation operations, brushing, filtering, and drag and drop, are normally implemented under the assumption that there is a many-to-one mapping between graphical objects and data objects. If graphical objects in multiple visualizations refer to the same data objects, these operations will affect all the visualizations in the same way, in which case they are said to be *coordinated.* Becker and Cleveland [3] originally conceived this architectural approach to coordination, and it represented a significant advance over earlier visualizations using ad hoc data structures. However their elegantly simple notion breaks down in the face of a database with multiple object types.

In previous data exploration systems using relational databases, the database contains a table of houses and a table of sales. Figure 1 was generated entirely from the house table, so its graphical objects are mapped to records in that table. To graph selling_price versus lot_size in Figure 2, we must form a relational join between the two tables, generating a third table whose attributes include both selling_price and lot_size. Graphical objects in the chart map to records in the new table. Therefore the two visualizations cannot be coordinated under Becker's architecture. Further, records in a relational database table have no structure beyond its set of attributes. They contain no memory of what tables they were generated from. So even with a more complicated coordination architecture, there is just not enough information in table records to do coordination based on object identity.

Visage solves this problem within the information-centric paradigm by mapping graphical objects to graph structures of objects, which are called *threads.* Because threads remember the objects from which they are constructed, coordination across visualizations with related but distinct thread structures is possible (see section 4). Threads provide an architectural solution to the multiple-object problem, but not a UI solution. There is still the need for queries to specify how threads are constructed, as well as other dataset definition operations.

1.2 VQE

VQE is a visual query environment within Visage for representing operations explicitly. It enables an analyst to construct complex intentional queries during data exploration and reuse them later. VQE queries can express navigation across relationships, aggregation, and filtering by range selection or by arithmetic or equality relations between attributes.

In traditional relational database query languages, queries are applied to tables in databases, generating new tables. As we have seen, the problem with this is that tables are at the wrong level of granularity. In order to link visualizations from multiple queries, visual objects should represent meaningful objects such as houses, rather than records from arbitrary tables. Therefore in VQE queries

are applied to threads in databases, generating new threads. Since VQE is based on threads, queries are fully integrated with the rest of Visage: not only can query results be dragged to other visualizations, but objects from visualizations can be dragged into queries (i.e. be the input to query expressions). User directed changes to queries and visualizations are immediately reflected in each other.

In summary:

- Queries can be created by drag and drop operations from visualizations, and visualizations can be created from queries.
- Queries are expressed using direct manipulation of a visual language.
- All visualizations created during an exploration session remain coordinated as the query is modified.
- Objects of one type can be filtered based on attributes of objects of another type.
- Visualizations can combine attributes from multiple object types.
- An exploration session consisting of queries and visualizations can be saved independently of any data, and reused on a different data set.
- Queries can express aggregation, equality and inequality constraints among attribute values, arithmetic expressions, navigation, and filtering. (Only the latter three are described in this paper.)
- A related tool allows browsing and editing the database schema.

2. EXAMPLE

Visualization and querying, both extensional and intentional, are illustrated on a fictitious database that might have been collected by a group of real-estate agents describing clients and sales information for three neighborhoods for 1989. The database schema is shown in Figure 2. The agent has a client who would prefer to live in Point Breeze, but is willing to consider Squirrel Hill since the average price of houses there is lower. The agent plans to explore whether the mere fact of the neighborhood label has an effect on price. Pittsburgh is famous for the character of its neighborhoods, and boundaries between them are labeled on street signs. However there are no major barriers separating these neighborhoods, and the character of the houses does not change abruptly. If houses just over the border in Squirrel Hill are less expensive, the client may be better off buying there.

2.1 Intentional Object Sets

The agent would like to see a map like Figure 1 that also encodes price. Unfortunately, price is not an attribute of houses, so a query is required. In Figure 1, the user has selected some houses that straddle the border with a bounding box. He copy-drags this set to VQE in order to

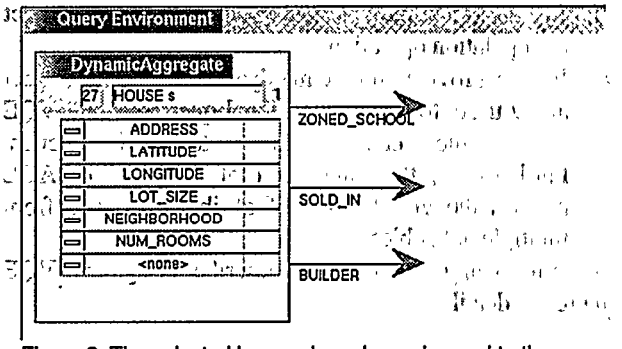

Figure 3: The selected houses have been dragged to the QueryEnvironment frame (VQE), where they become a dynamic aggregate. The arrows to the right constitute a menu for parallel navigation.

begin query construction (Figure 3). The set of houses is visualized as a *dynamic aggregate*. It is an aggregate because operations on it apply to each of its 27 elements; it is dynamic because it represents an intentional set. As the user changes the query, the set of houses that satisfy it (its extension) changes. Even if the query doesn't change, it may be applied to a different set of houses simply by dragging objects into or out of dynamic aggregates.

2.2 Intentional Navigation

The agent first wants to link the houses with their sales, so has brought up a menu of relationships that houses participate in. He will select the sold_in relationship and drag its arrowhead to a screen location where a second dynamic aggregate will be placed (see Figure 4, top). We call this operation *parallel navigation*, because we are navigating across the sold_in relationship for each element of the aggregate. The new dynamic aggregate will be the aggregate of all the sales of these houses. Parallel navigation corresponds to a join in a relational query, or a path expression in an object-oriented query. If objects are added to or removed from a dynamic aggregate, the intentional navigation representation allows other linked dynamic aggregates to update, maintaining query consistency.

2.3 Creating Visualizations

Now that the houses and sales have been linked, the agent can specify how to encode attributes from both types of objects within the same visualization. He constructs a SageBrush sketch (Figure 4, middle) for a map just like Figure 1, except that it will encode selling price by the size of the points. Figure 4 (bottom) shows the result. Each visual object on the map now represents a <house, sale> thread, but coordination still happens via the thread objects.

The new map does little to confirm the neighborhood label hypothesis: the variance in point size overwhelms any systematic difference, which if anything favors Squirrel Hill with the expensive houses.

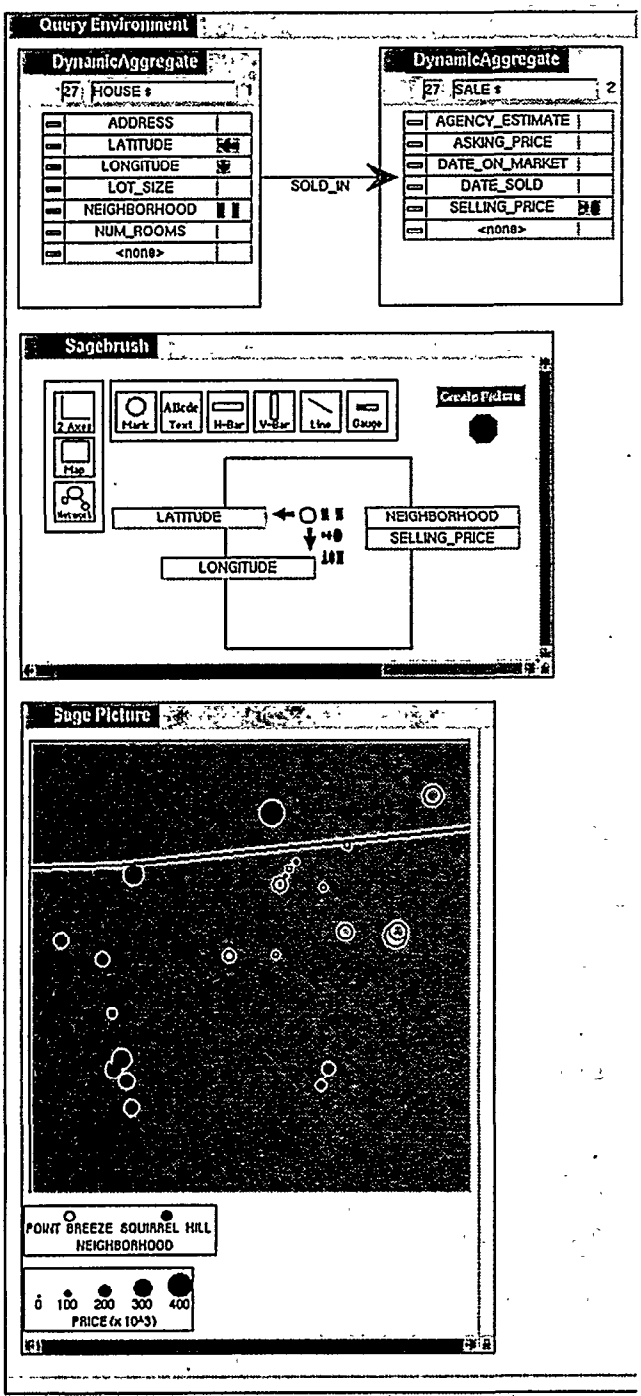

Figure 4: Middle: A SageBrush sketch calling for a map with points inside it. Attributes from the dynamic aggregates have been mapped to graphical properties of the point. Bottom: The resulting visualization.

2.4 Defining New Attributes

A second question is whether the client will be able to qualify for a higher mortgage in one neighborhood than another. Although this information can't be predicted with certainty, an indicator is the ratio of selling_price to buyer salary for other sales in the two neighborhoods. This ratio is not defined for any of the object types in the database, and so the only recourse in previous

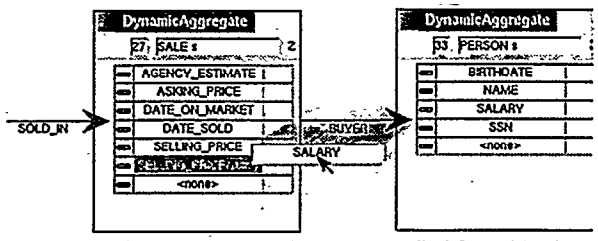

Figure 5: Defining the new attribute to be called StretchRatio.

visualization systems would be another session with the database to generate another table. VQE provides a spreadsheet-like interface for this purpose. First the agent performs another parallel navigation from sale across the buyer relationship to person, in order to access the salary attribute. Figure 5 illustrates the definition of StretchRatio as selling_price/sum(buyer→salary). That is, for each sale a new attribute value is computed by navigating to all the buyers in the sale, adding up their salaries, and dividing into the selling price. The new attribute is a permanent addition to the database schema, and may be used in queries that don't involve a person dynamic aggregate.

Figure 6 shows a new map where size encodes StretchRatio rather than selling_price. As a memory aid, the agent has edited the title of the visualizations to "SellingPrice" and "StretchRatio." The variance has been reduced, and a systematic tendency to buy a more expensive house for a given salary level is apparent. The client's original preference for Point Breeze is well supported. There is no indication that she will pay more for the same quality house, and she may be able to borrow more and therefore afford a nicer house.

2.5 Extensional Query Update

Having found an interesting relationship between Squirrel Hill and Point Breeze, the agent wonders whether it applies to Squirrel Hill and Shadyside. The exploration sequence can be reused simply by dragging more objects into the visualization. Using another bounding box on the map of Figure 1, the agent picks out houses near the border between Squirrel Hill and Shadyside, and copy drags them to one of the maps inside VQE. Recall that points on the original map represented houses, while those on the second map represent <house, sale> threads, and those on the third represent <house, sale, person> threads. When the drop occurs, Visage consults the thread representation to perform the appropriate navigation operations necessary to convert from one thread structure to another (see section 4). In this way, the appropriate objects are added to the two visualizations and the three dynamic aggregates (Figure 7).

VQE queries have both intentional and extensional aspects. The graph structure and slider settings form the

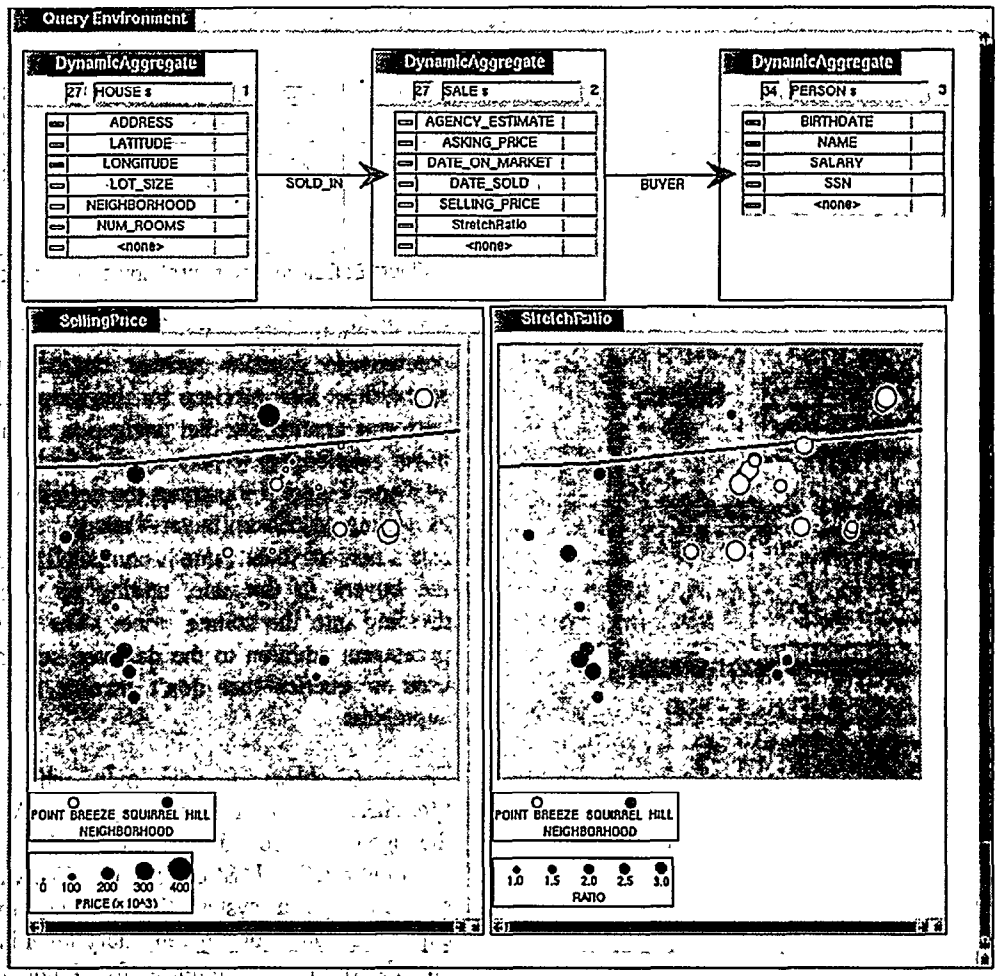

Figure 6: The new visualization (lower right) shows the StretchRatio of the border houses.

intention, which is dynamically applied to any objects extensionally dragged in or out of any dynamic aggregate or visualization.

2.6 Saving the Query

Just as easily as new objects were added to the query, the old objects can be removed, leaving a bare query that can be named and saved for reuse in a later session. We intend to extend the SageBook [5] interface to enable searching for queries based on characteristics of the visualizations or the query.

2.7 Different Input to the Query

Later, the agent has been exploring pricing strategies. Is it better to ask a high price and be willing to negotiate more, or to offer the lowest price up front? In Figure 8, the agent has defined two new attributes: Days_on_Market = date_sold - date_on_market is displayed on the x-axis; %_Price_Drop = 100*(1 - selling_price/asking_price) is displayed on the y-axis. Interestingly, there is little correlation. Some houses even sell for more than the asking price, including one outlier that increases almost 60%! This last is probably a data entry error.

Out of curiosity, he wonders whether %_Price_Drop has any impact on the StretchRatio location dependence found earlier. He finds and restores the earlier exploration session, and copy drags all the points from the chart onto the sale dynamic aggregate (Figure 9). VQE navigates to the appropriate houses and buyers, and populates the two maps. Note that the navigation is now being done in a different direction, because we are starting with sales rather than houses. This is possible because the query is a declarative specification of constraints among objects, rather than a procedure as might be captured by a programming-by-demonstration macro facility.

2.8 Brushing and Filtering

Figure 9 also shows several further exploration operations. First, the agent has brushed the highest price drops in the chart with black paint, causing the corresponding houses to be painted black on the maps. He has also dropped dynamic query sliders on several of the attributes. These sliders include histograms showing the distribution of values for the attribute. The rectangular outlines show the currently selected range for each slider. Only the range of the lot_size slider has been restricted, to filter out houses

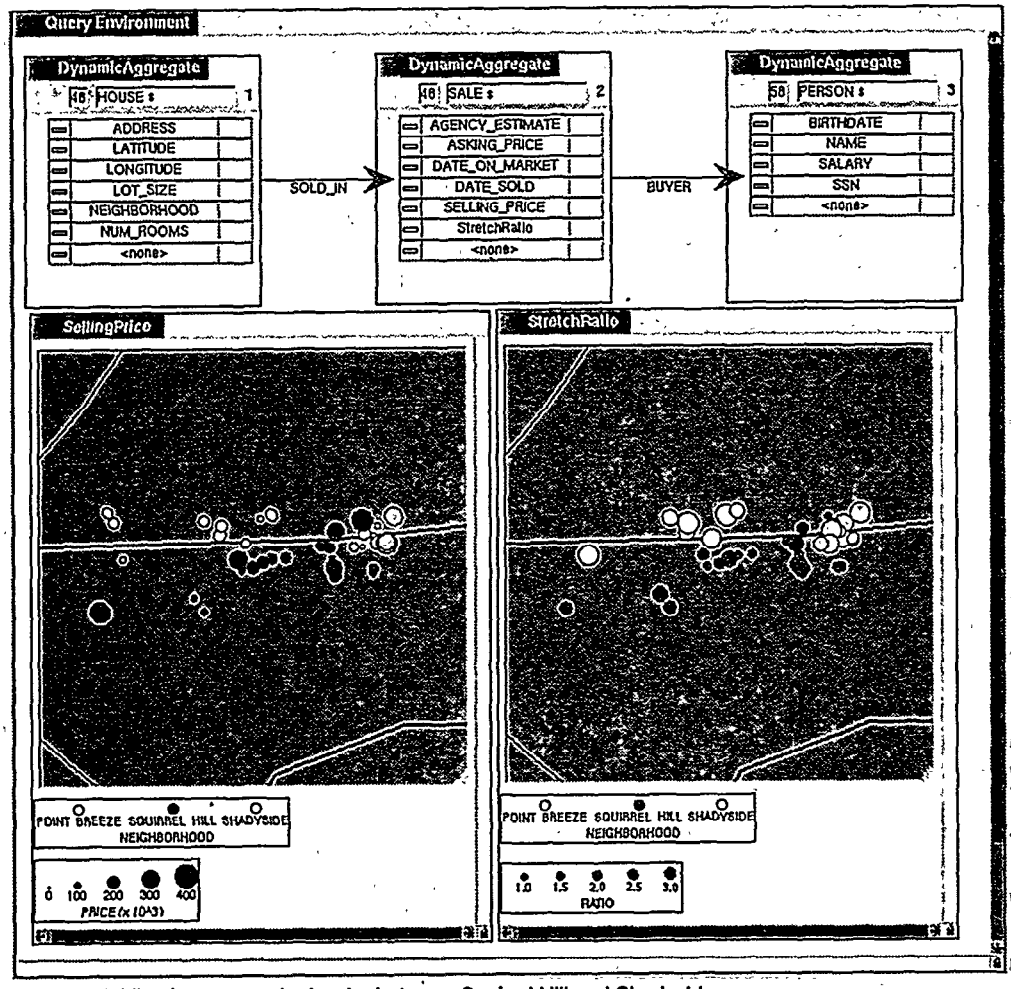

Figure 7: Adding houses on the border between Squirrel Hill and Shadyside.

with small lots. The histograms are updated to show the distribution of filtered values in black, while the original distribution remains in gray.

3. USEFULNESS OF THREADS

Direct manipulation interfaces are easy to learn and use at least in part because they map interface objects onto the user's intuitive object-oriented domain model. For applications where relationships among multiple objects are important, previous architectures for direct manipulation do not apply. We are unaware of any previous use of composite first class objects like threads, which enable the interface to transparently respect object identity while manipulating attributes of distinct but related objects.

Even within the object-oriented database community, there is no mention of first class objects that refer to structures of domain-level objects. Object-oriented databases include *path expressions* [13] for describing navigation operations, but they depend on a host programming language to extract attributes of related objects. OQL query semantics are defined in terms of *extensional patterns* [2], which are like threads, but are merely theoretical pedagogical concepts.

Not providing threads as first class objects is most likely for efficiency reasons. Since tuples of objects can be reconstructed at will, what is the point of maintaining them in persistent store rather than ephemerally in the host language's variable bindings or other data structures?

We argue that the overhead is justified by the increased architectural modularity afforded the interface. As GUI's become ever more complex, there is a trend to make them application-independent utilities. Visage takes advantage of this separation in order to present multiple applications to the user as one, with transparent coordination and communication between them, and consistent behavior across them. This separation requires that all data be stored in a consistent form in a shared database, however. Anything required by the interface must be in this persistent store.

4. IMPLEMENTATION

Visage is implemented on top of the Saga GUI toolkit from Maya Design Group, which runs on Macintosh, IBM PC, and Unix/X. VQE is implemented on top of Visage,

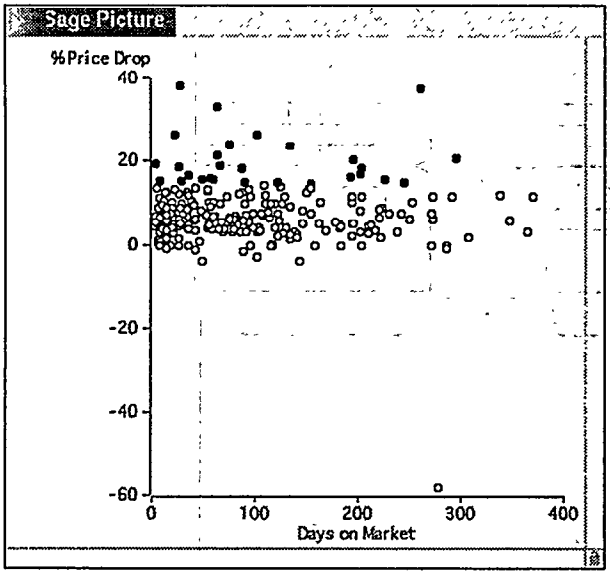

Figure 8: Percentage drop from asking_price to selling_price is plotted against the number of days on the market. Sales with a large drop in price have been brushed with black paint.

using Saga's scripting language. Below are brief descriptions of algorithms for extending direct manipulation operations to deal with threads.

Brushing. When a graphical object is brushed, a message is sent to all visualizations to color corresponding objects with the current brushing color. This is just as in any brushing algorithm. The hitch is that in VQE we want brushing to be coordinated across visualizations generated from multiple queries, which may have different thread structures. So we send a separate message for each object of the thread. A visualization colors a thread if any of its objects match the argument of the message.

For instance, when the high-drop sales are brushed in Figure 8, and the visualizations in Figure 9 are given the message, they both find graphical objects to color, because their threads contain sale objects. On the other hand, the map in Figure 1 does not color any of its graphical objects, which mention only houses. In contrast, if the user had painted objects in the SellingPrice visualization in Figure 9, messages would have been sent to color houses, and more messages would have been sent to color sales. Therefore objects in Figure 1 (as well as in Figures 8 and 9) would have been colored.

Dynamic Query. Each query graph maintains a list of threads that constitute the current dataset to which the query is applied. For each thread it remembers how many DQ sliders are currently filtering out the thread. Each DQ slider is associated with a node in the query graph and an attribute. Its job is to tell the query when it filters out or reinstates a thread. When it receives the message, the query increments or decrements the counter associated with the thread. If the count decreases to zero or increases from zero, the visibility of all objects representing the thread must be changed and the histograms of all sliders must be updated. So far, the algorithm is identical to DQ algorithms for ordinary objects. Again we deal with the difference between threads and ordinary objects by issuing a separate visibility message for each thread object.

Updating histograms is more complicated than in the non-thread case, because they record the frequency of objects rather than threads. An object is considered to be filtered out iff all threads containing it are filtered out. Therefore each query node must maintain a count of the number of threads filtered out for each object. When this number crosses a threshold, the histogram is updated just as in the ordinary case [14].

Drag and Drop. Rather than storing the graph structure on every thread, it is stored as a separate template, and all threads with the same structure point to the same template. The template records the object type of each node, and the relationships between pairs of nodes. Every Sage visualization is designed to display threads with a particular template. When dropping a graphical object into a visualization, Visage tries to coerce the corresponding thread into a thread with that visualization's template. (Visage treats ordinary objects as unary threads, so conversion among threads covers the general case.) Thread conversion consists of two steps. First, the template of the source thread must be mapped to that of the destination thread. All possible one-to-one mappings from subsets of the source nodes to subsets of the destination nodes are considered. Second, a destination thread is created and its nodes are bound to data objects. For each possible mapping, the data objects bound to the mapped nodes in the source thread are copied to the destination thread. Other nodes on the destination thread, if any, are initially empty. By depth-first search, the empty nodes are populated with any data object of the correct type. If at any point there are two thread nodes connected by a relationship that doesn't hold among the objects bound to those nodes, the search backtracks. (For efficiency, candidate data objects can be generated by navigation along the relationships, rather than by the object type.) All complete destination threads are added to the destination visualization.

5. RELATED WORK

5.1 GQL

GQL [9] is a fully visual conceptual level Graphical Query Language with the expressiveness of SQL. VQE's visual representation of the query graph is adapted from GQL, with some interface modifications such as using containment to show attributes rather than links. The

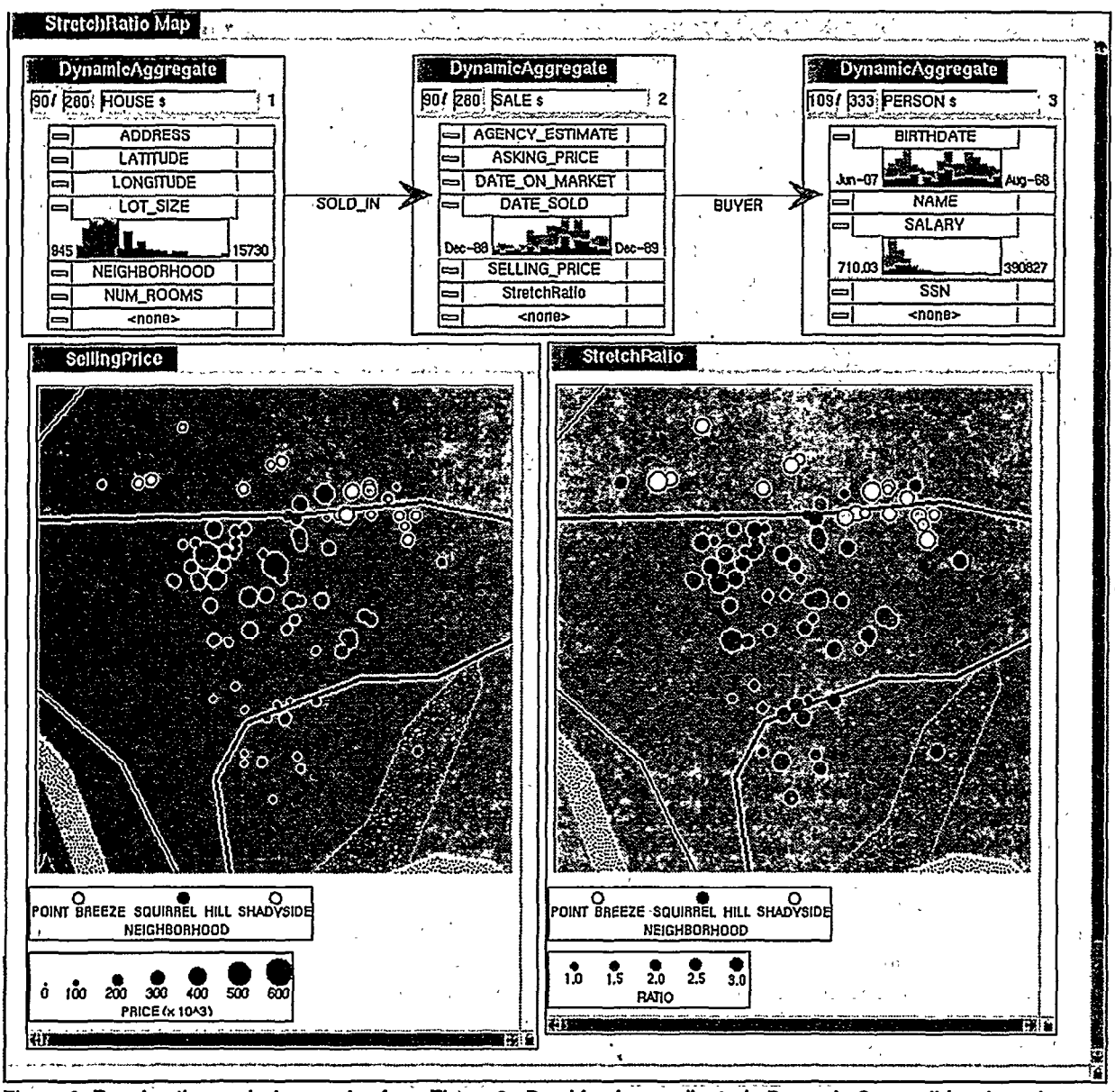

Figure 9: Reusing the analysis on sales from Figure 8. Brushing is coordinated. Dynamic Query sliders have been added.

problem with GQL as it stands is that it is not integrated with a visualization system for displaying query results. Each query generates a static table, so the paradigm is batch processing rather than the incremental query and direct manipulation exploration of VQE. There are many other conceptual level query interfaces from the database community, but all are less elegant than GQL, and none visualize results in an interesting or incremental way (see for instance many of the papers in [6]).

5.2 Exbase

Exbase [7] is similar to this work in terms of motivation, in that it seeks to provide an intermediary between a database and a visualization system. However the emphasis is on explicitly representing the history of user interaction with the database and visualizations. Lee and Grinstein distinguish remote database accesses from local processing, so there are objects for database views, which are the result of queries, and derived views, for example as the result of manipulating sliders. Similarly, they maintain a derivation history of visualization views.

We have not yet addressed the question of maintaining histories, having chosen to focus on intuitive query languages and integration of querying and visualization - topics that Exbase, which uses SQL as a query language, has not yet addressed.

5.4 Butterfly

Butterfly [8] is a citation searching and browsing application. Searching is done by selecting databases and attribute/value pairs. Browsing is done by following reference and citer links. Butterfly is able to hide the complex database queries from the user, largely because

the class of queries can be anticipated. To complete the analogy with VQE, Butterfly would be able to modify queries by operations on query results. For instance, additional keywords associated with a citation might be dragged into the query.

5.5 IMACS

IMACS [4] has very similar goals to Visage/VQE. It supports analysts' iterative exploration and visualization of data, including query reuse. It's foundation is the knowledge representation system CLASSIC, which is much more sophisticated than the object-oriented database we use. The primary advantage of this is that subsumption relations among queries can be inferred, allowing them to be automatically organized into the knowledge base. On the other hand, in later work the same group reverted to a simpler knowledge representation system to decrease the overhead and allow exploration of larger data sets [12].

IMACS visualizations do not support direct manipulation interaction, and it uses a textual query language. Queries are somewhat more expressive than VQE's.

6. SUMMARY

VQE combines a GQL-style intentional visual query language with direct-manipulation data exploration capabilities as found in systems like Visage, IVEE, and the Influence Explorer. Since queries and visualizations share an object oriented database, visualizations can combine attributes of multiple objects, and visualizations resulting from a sequence of queries are coordinated. Integration of extensional and intentional exploration allows use of direct manipulation where possible but still retains the ability to capture and reuse query sequences as declarative structures. VQE frames containing a sequence of nodes and links and associated visualizations can be saved and/or cleared of data to be reused with new datasets.

These capabilities are supported by the concept of threads, which we believe will be generally useful as interfaces become more application-independent, and support inter-application communication via shared object-oriented databases.

REFERENCES

[1] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '92)*, pages 619-626. ACM Press, 1992.

[2] A. Alashqur, Stanley Su, and H. Lam: OQL: A Query Language for Manipulation Object-oriented Databases. In *Proceedings of the 15th International Conference on Very Large Data Bases*, 1989, pages 433-442.

[3] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2), 1987.

[4] R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuinness, and L. A. Resnick. Integrated support for data archaeology. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):159-185, 1993.

[5] M. C. Chuah, S. F. Roth, J. Kolojejchick, J. Mattis, and O. Juarez. Sagebook: Searching data-graphics by content. In Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '95), pages 338-345. ACM/SIGCHI, 1995.

[6] R. Cooper, editor. *Proceedings of 1st International Workshop on Interfaces to Database Systems*. Springer-Verlag, 1993.

[7] J. P. Lee and G. G Grinstein. Describing visual interactions to the database: closing the loop between user and data. In *Proceedings of Visual Data Exploration and Analysis III (SPIE '96)*, 1996.

[8] J. D. Mackinlay, R. Rao, and S. K. Card. An organic user interface for searching citation links. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '95)*. ACM Press, 1995.

[9] A. Papantonakis and P. J. H. King. Syntax and semantics of GQL, a graphical query language. *Journal of Visual Languages and Computing*, 6:3-25, 1995.

[10] S. F. Roth, M. C. Chuah, S. Kerpedjiev, J. A. Kolojejchick, and P. Lucas. Towards an information visualization workspace: Combining multiple means of expression. *Human-Computer Interaction*, in press, 1997.

[11] S. F. Roth, P. Lucas, J. A. Senn, C. C. Gomberg, M. B. Burks, P. J. Stroffolino, J. A. Kolojejchick, and C. Dunmire. Visage: A user interface environment for exploring information. In *Proceedings of Information Visualization*, pages 3-12. IEEE, 1996.

[12] P. G. Selfridge, D. Srivastava, and L. O. Wilson. Idea: Interactive data exploration and analysis. In *Proceedings of SIGMOD 1996*, 1996.

[13] Z.-H. Tang, G. Gardarin, and V. Smahi. Optimizing path expressions using navigational algebraic operators. In Proceedings of Database and Expert Systems Applications, DEXA '96, pages 574-583, 1996.

[14] L. Tweedie, R. Spence, H. Dawkes, and H. Su. Externalising abstract mathematical models. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '96)*, pages 406-412. ACM/SIGCHI, 1996.

NITELIGHT: A Graphical Tool for Semantic Query Construction

Alistair Russell
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
ar5@ecs.soton.ac.uk

Paul R. Smart
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
ps02v@ecs.soton.ac.uk

Dave Braines
Emerging Technology Services
IBM United Kingdom Ltd,
Hursley Park, Winchester,
Hampshire,
SO21 2JN, UK
dave_braines@uk.ibm.com

Nigel R. Shadbolt
School of Electronics and Computer Science
University of Southampton
Southampton
SO17 1BJ, UK
nrs@ecs.soton.ac.uk

ABSTRACT
Query formulation is a key aspect of information retrieval, contributing to both the efficiency and usability of many semantic applications. A number of query languages, such as SPARQL, have been developed for the Semantic Web; however, there are, as yet, few tools to support end users with respect to the creation and editing of semantic queries. In this paper we introduce a graphical tool for semantic query construction (NITELIGHT) that is based on the SPARQL query language specification. The tool supports end users by providing a set of graphical notations that represent semantic query language constructs. This language provides a visual query language counterpart to SPARQL that we call vSPARQL. NITELIGHT also provides an interactive graphical editing environment that combines ontology navigation capabilities with graphical query visualization techniques. This paper describes the functionality and user interaction features of the NITELIGHT tool based on our work to date. We also present details of the vSPARQL constructs used to support the graphical representation of SPARQL queries.

Author Keywords
sparql, visual query system, semantic web, graphical query language, ontology, owl.

ACM Classification Keywords
graphical user interfaces, query formulation, query languages

INTRODUCTION
Information retrieval is a key capability on the Semantic Web, contributing to both the efficiency and usability of many semantic applications. The availability of semantic query languages such as SPARQL [20] is an important element of information retrieval capabilities; however, query developers are likely to benefit from the additional availability of tools that assist them with respect to the process of query formulation (i.e. the process of creating or editing a query). Ideally, query formulation tools should avail themselves of user interaction capabilities that contribute to the efficient design of accurate queries while maximally exploiting the power and expressivity provided by the constructs of the target query language.

Most attempts to support the user with respect to query formulation have focused on graphical or visual techniques in the form of Visual Query Systems (VQSs) [7]. VQSs provide a number of advantages relative to simple text editors. Most obviously, such systems support the user in developing syntactically valid queries: they serve to constrain or guide editing actions so as to militate against the risk of lexical or syntactic errors. Other potential advantages include improved efficiency, understanding and reduced training requirements.

In this paper we introduce a graphical tool for semantic query construction based on the SPARQL language specification [20]. SPARQL is one of a number of query

languages that have been proposed for the Semantic Web. Others include RQL [14] and RDQL [22], although only SPARQL benefits from W3C endorsement. The tool we present in this paper (called NITELIGHT) enables users to create SPARQL queries using a set of graphical notations and GUI-based editing actions. The tool is intended primarily for users that already have some familiarity with SPARQL; the close correspondence between the graphical notations and query language constructs makes the tool largely unsuitable for users who have no previous experience with SPARQL.

The rest of this paper is organized as follows. We first provide an overview of the SPARQL query language. The purpose of this overview is to highlight the target set of constructs that need to be supported by any (fully) SPARQL-compliant Visual Query Language (VQL). The following section (Graphical Query Editor) describes the NITELIGHT tool we have developed to support graphical query formulation. We first present the graphical notations that comprise the elements of the VQL supported by NITELIGHT (a language we refer to as vSPARQL); we then go on to describe the tool itself, describing both its general functionality and support for user interaction. Next we present previous work in the area of graphical query formulation, particularly in the context of the Semantic Web. The emphasis in this section is, not surprisingly, on graphical techniques, particularly those provided by Visual Query Systems (VQSs); however, we also describe approaches based on natural language interfaces. Finally, we describe some directions for future work based on our progress to date.

SPARQL QUERY SYNTAX

SPARQL [20] is a semantic query language that exploits the triple-based structure of RDF to perform graph pattern matching and contingent RDF triple assertion. In this sense it is similar to RDQL [22]; however, SPARQL provides a number of features that are not provided by RDQL [see 4 for a review]. These include:

- the ability to create new RDF graphs based on query variable bindings (this is accomplished using the SPARQL CONSTRUCT form)
- the ability to return descriptions of identified resources in the form of an RDF graph (this is accomplished using the SPARQL DESCRIBE form)
- the ability to specify optional query graph patterns (this allows a user to specify that data should contribute to an answer if it is present in the RDF model)
- the ability to test for the presence or absence of specific triple or graph patterns via the SPARQL ASK query form

SPARQL includes facilities to filter result sets using specific tests, e.g. to test whether or not a particular query variable is bound or unbound. It also includes a number of solution sequence modifiers (ORDER BY, DISTINCT, OFFSET, LIMIT, etc.) that modify the sequence of query solutions returned by a SPARQL query processor. SPARQL is, in summary, a highly expressive semantic query language that compares favorably with other RDF query languages, such as RDQL and SeRQL [see 13]. Figure 1 and Figure 16 provide examples of SPARQL queries.

GRAPHICAL QUERY EDITOR

The development of a graphical tool for SPARQL query formulation necessarily entails the development of a set of graphic notations that support the visual representation of SPARQL query components. Following an analysis of the SPARQL syntax specification [20], we developed a set of graphical notations to support the representation of SPARQL queries. These notations comprise the basis of a SPARQL VQL that we refer to as vSPARQL. In the first half of this section we present some features of this language based on our work to date. The graphical query designer, NITELIGHT, was designed to support the user with respect to the formulation of SPARQL queries using vSPARQL constructs. The second half of this section describes the functionality and user interaction features of the NITELIGHT editor.

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX sem: <http://www.edefence.org/semiotiks.owl#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

SELECT ?incident, ?type, ?certainty
WHERE
{
    ?incident rdf:type sem:MineIncident .
    ?incident sem:isLocatedIn sem:HelmandProvince .
    ?incident sem:involves ?mine .
    ?mine rdf:type sem:Mine .
    ?mine sem:usesExplosive sem:TNT .
    ?mine sem:hasType ?type .
    ?mine sem:hasTypeP ?certainty .
    FILTER (?certainty > .50)
}
```

Figure 1. SPARQL SELECT Query

Graphical Notations

Because SPARQL queries exploit the triple-based structure of RDF models, graph-based representations comprising a sequence of nodes and links can be used to represent the core of most SPARQL queries, i.e. the basic triple patterns that are matched against the RDF data model. The nodes in this case correspond to the subject and object elements of an RDF triple; the links correspond to RDF predicates.

Basic Triple Patterns

In terms of the vSPARQL language, nodes and links correspond to URIs, literal values or variables (bound or unbound). Nodes are represented graphically as a geometric

object exploiting both color and shape to indicate the node type (e.g. unbound variable). Nodes are also associated with a label that indicates the URI, literal value or query variable represented by the node (see Figure 2).

Links are represented as simple lines. They are also associated with a label that indicates the predicate represented by the link or the name of a query variable. Directional arrows indicate which node represents the subject and which node represents the object in a triple pattern (see Figure 2).

Multiple Triple Patterns

The introduction of multiple triple patterns into a query is represented by the addition of multiple nodes and links (see Figure 3). If there are any shared variables or literal values across the triple patterns, then these are represented using a common graphical node with multiple link connections.

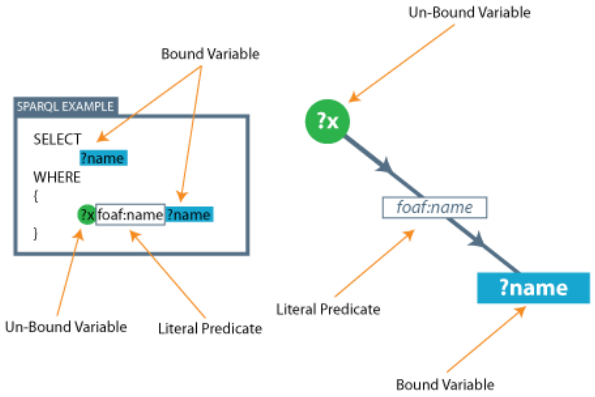

Figure 2. Basic Triple Pattern

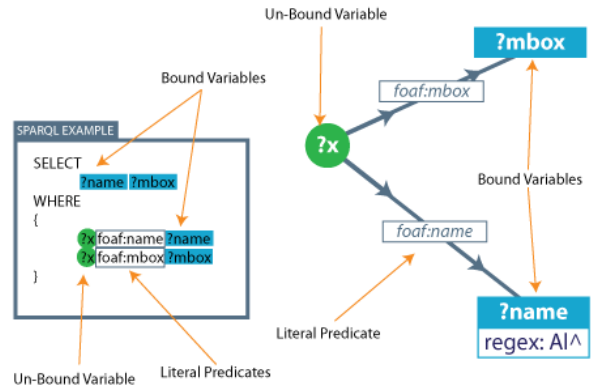

Figure 3. Multiple Triple Patterns

Variable and Triple Ordering

For some SPARQL queries, the ordering of triple patterns and bound variables is important. In order to support the user with respect to the ordering of variables and triple patterns, a numeric value is displayed in the top left corner of both node and link labels (see Figure 4). Any nodes that are duplicated across graph pattern groups will share the same order indicator.

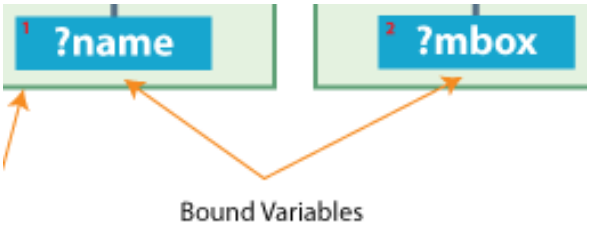

Figure 4. Variable and Triple Ordering

Graph Patterns

In SPARQL, a graph pattern consists of one or more triple patterns that are matched against the entire RDF graph. Graph patterns influence variable bindings because each variable has local scope with respect to the graph pattern in which it is contained. This means that the same variable could be bound to different values in different graph patterns. Using graph patterns means that the triples within a graph pattern are matched against the entire RDF graph and are not affected by any previous graph patterns.

Graphical support for the representation of graph patterns in vSPARQL is accomplished by organizing node-link-node collections into groups (see Figure 5).

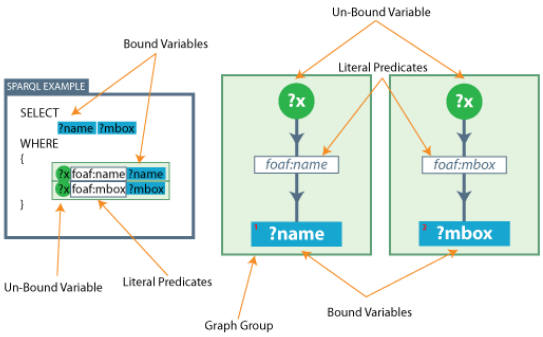

Figure 5. Graph Patterns

When shared nodes appear in multiple graph patterns, the nodes are duplicated graphically. Internally, however, duplicated nodes are treated as the same node.

Optional Graph Patterns

The representation of optional graph patterns is accomplished by visually highlighting the relevant triple groups within the optional graph pattern (Figure 6).

Union Graph Patterns

The visual representation of union graph patterns (i.e. graph patterns where either one of two graph patterns could be considered as part of a query solution) is accomplished by

linking two graph pattern groups with a union label indicator (see Figure 7).

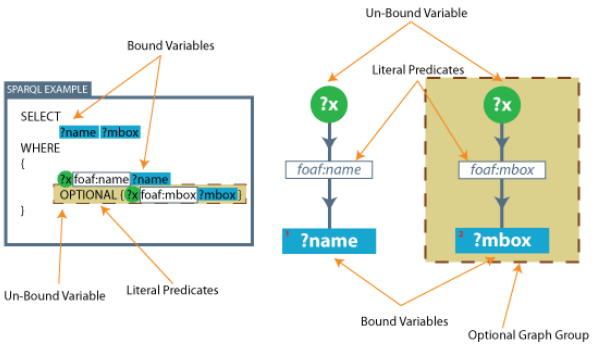

Figure 6. Optional Graph Patterns

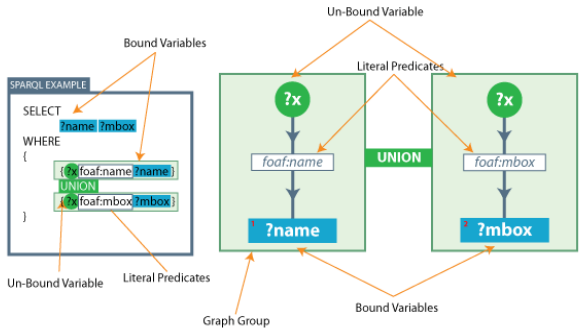

Figure 7. Union Graph Patterns

Graph Specification

The specification of a default RDF graph, or the retrieval of a graph as part of a query, is represented by assigning a variable or literal value to a graph pattern group (see Figure 8).

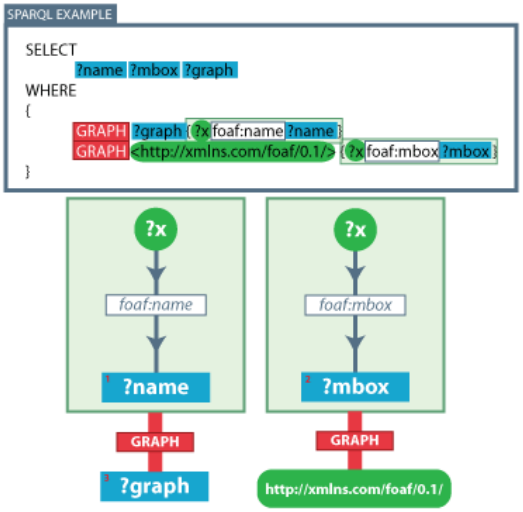

Figure 8. Graph Specification

Result Ordering

SPARQL query results can be ordered by any variable, bound or unbound. To represent this visually, a numerical indicator, similar to the variable/triple pattern order number indicator (see Figure 4), is used. In this case the numeric value appears on the right-hand-side of the variable nodes. The graphical indicator also provides information about the sort order (i.e. ascending or descending) using a directional arrow (see Figure 9). If no indicator is present, the variable is not used for the purposes of ordering the query result set.

Figure 9. Result Ordering

Variable Filter

SPARQL filtering is used to restrict the result sets returned by a query using numerical and regular expressions. The visual representation of a filter expression is based on the addition of a filter field box to the node or link that participates in the filter expression (see Figure 10).

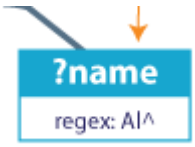

Figure 10. Filter Expressions

Distinct, Limit and Offset

SPARQL also provides functions for retrieving distinct result sets, as well as limiting result sets to a specified number of solutions. These functions are all global to the current query, and can be viewed or changed using the GQE.

Query Editor Prototype

To test and evaluate the features of vSPARQL, we developed a Java-based prototype application, called NITELIGHT, using a combination of Jena [19] and Standard Widget Toolkit (SWT) components.

NITELIGHT (see Figure 11) provides 5 distinct components, each of which works together to give the user an intuitive interface for graphical query creation.

The centerpiece of the NITELIGHT tools is the Query Design Canvas (see Figure 12). The functionality of this component is supplemented by an Ontology Browser component (see Figure 13), a SPARQL Syntax Viewer, a Query Results Viewer and a Quick Toolbar.

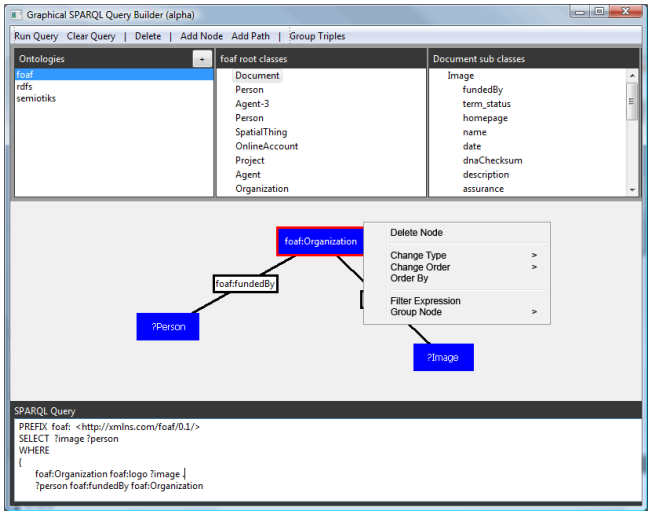

Figure 11. Query Editor Prototype Interface

Query Design Canvas

The Query Design Canvas (see Figure 12) is the centerpiece for user interaction and query construction in the NITELIGHT tool. It provides a canvas for the graphical rendering of SPARQL queries using vSPARQL constructs. It also includes a number of user interaction features that allow users to create and refine semantic queries.

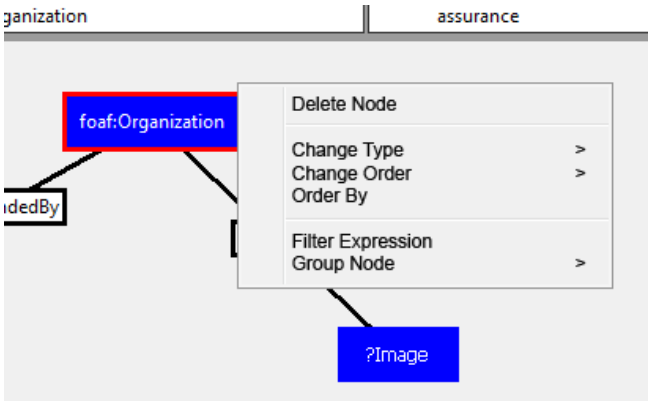

Figure 12. Query Design Canvas

Triples are drawn as two polygon nodes joined with a single link. To allow for more complex queries, the polygon nodes can be moved around the canvas freely, and the canvas itself can be zoomed and panned to view the entire query at different levels of visuo-spatial resolution.

Both the nodes and links are selectable objects that can be edited using either the Quick Toolbar or a context menu. Both the Quick Toolbar and the context menu allow users to define filtering, ordering and grouping information for the selected object. The support for defining filter expressions is currently limited, consisting of a simple text entry form. Our future development plans aim to provide better support for filter expression definition, perhaps using a wizard-like utility.

Ontology Browser

To facilitate the process of query formulation, and to provide users with a starting point for query specification, the NITELIGHT editor includes an Ontology Browser component (see Figure 13). The first column of the Ontology Browser is a persistent list of currently loaded ontologies (the Source Ontologies Column). New ontologies can be loaded into the browser, and the selection of one of the loaded ontologies will result in the enumeration of top-level classes (root classes) in the second column of the Ontology Browser.

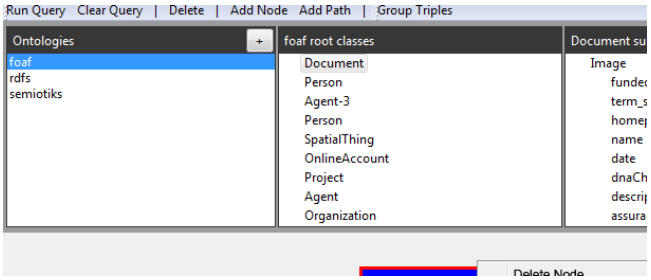

Figure 13. Ontology Browser

The Ontology Browser consists of a series of columns that display the classes and subclasses of an ontology with more abstract classes situated to the left. The column immediately to the right of the Source Ontologies Column is always populated with the root classes of the currently selected ontology. Selecting a class from this column causes an adjacent column to appear to the right of the root classes column. This new column contains the subclasses of the currently selected root class. The pattern of subclass enumeration is repeated as the user progressively selects classes from the right-most column.

The Ontology Browser also provides access to information about the properties associated with each class. In this case, the user can expand a class node in the Ontology Browser to view a list of properties associated with the class.

The Ontology Browser enables a user to drag and drop classes and properties onto the Query Design Canvas. A new node can be created by dragging a class item from the Ontology Browser onto the canvas. A new link can be created by dragging a property from the Ontology Browser and attaching it to a node on the canvas.

SPARQL Syntax Viewer

The SPARQL Syntax Viewer component provides a text-based view of the query that is dynamically updated to reflect any changes made using the Query Design Canvas. At the present time, the SPARQL Syntax Viewer is read-only, i.e. the user cannot edit the SPARQL syntax directly; they must implement any changes to the query via the Query Design Canvas. Future work could explore the possibility of bi-directional translation capabilities in which the user would be permitted to modify the graphical representation of a SPARQL query by interacting directly with the SPARQL Syntax Viewer. This would be of

particular benefit to users who wanted to visualize existing text-based SPARQL queries for the purposes of query refinement or improved understanding.

Query Results Viewer

The Query Results Viewer allows a user to execute a vSPARQL query against any SPARQL endpoint. In the current version of the tool the results are presented in the form of a simple table; however, one could imagine a variety of alternative output formats that might be more suited to the processing capabilities of human end-users. Examples include map-based visualizations, timelines and natural language serializations of query result sets. Since these output formats are often tied to a particular application context, we do not intend to explore the use of these richer visualizations as part of the current development effort.

Quick Toolbar

The Quick Toolbar provides access to commonly used tools for manipulating the Query Design Canvas and its graphical query contents. Example tools include pan and zoom buttons, grouping functions and node editing utilities.

RELATED WORK

A number of approaches to query formulation have been described in the literature. This section provides an overview of some of the approaches that are related to the work described in this paper, or that impact on future extensions to the NITELIGHT query designer tool.

Visual Query Systems

Most attempts to support the user with respect to query formulation have focused on graphical or visual techniques in the form of VQSs [7]. VQSs are systems that use visual representations to depict the domain of interest and express related queries. Often they provide a language, a VQL, which defines both a set of graphical notations to represent query constructs and a compositional semantics for using the notations in the context of query formulation.

Perhaps the best known example of a VQS is the Query-By Example (QBE) system that was developed by IBM in the 1970s [23]. Since then many VQSs have been developed. Catarci et al [7] present a classification scheme for VQSs based on the kind of visual formalism (see [11]) used for query representation. They identify 4 categories of VQSs:

1. form-based systems: these are systems that provide structured representations corresponding to conventional paper-based forms. The aforementioned QBE system was one of the first systems to adopt a form-based approach.
2. diagram-based systems: these are systems that depict relationships between components using simple geometrical figures, such as squares, rectangles, circles, etc. Typically, a diagram-based system will use visual components that have a one-to-one correspondence with specific concepts, with lines between the components representing logical relationships between the concepts.
3. icon-based systems: these are systems that use icons to represent the concepts defined in the domain of discourse. Iconic representations have the advantage that they serve as a pictorial or metaphorical reminder of the concepts being represented; however, VQSs often need to represent entities that have no natural visual counterpart, e.g. an action, command or design specification.
4. hybrid systems: these are systems that comprise two or more of the aforementioned categories.

Of these systems, diagram-based systems tend to be the most popular. In fact, the tool we describe in this paper belongs to this particular category of VQS.

There have been a number of previous attempts to support graphical modes of query formulation in the context of the Semantic Web. Notable examples include OntoVQL [9], SEWASIE [8], SPARQLViz [6], and iSPARQL [2]. OntoVQL [9] is a graphical query language for OWL DL ontologies that maps onto the query language supported by the DL reasoner, Racer. One problem with OntoVQL concerns its expressive power, which is somewhat limited compared to conventional semantic query languages, such as SPARQL. In addition, there is, as yet, no one-to-one correspondence between the visual components of OntoVQL and the elements of a textual query language. This makes OntoVQL somewhat unsuitable as a graphical representational language for SPARQL.

SEWASIE [8] is a graphical query generation environment that co-opts natural language representations and graph-based visualizations of the domain ontology. The user is able to extend and customize an initial query by adding property constraints to selected classes or by replacing classes in the query with another compatible class, such as a subclass or superclass. This process of query refinement is accomplished by selecting terms in the sentential structure of a text-based representation of the query, and then interacting with a graphical visualization of a relevant part of the ontology infrastructure. As the user selects different parts of the query sentence, the graphical visualization of the ontology fragment is updated to reflect the kinds of editing actions that may be performed.

SPARQLViz [6] is a plugin for IsaViz [1] that provides a GUI for the graphical construction of SPARQL queries. SPARQLViz aims to support the user with respect to query formulation, and its aims are therefore similar to those of the work described herein. Significant differences emerge, however, in terms of the approach to user interface design. SPARQLViz relies on a wizard-like interface that presents the user with a sequence of forms such as that presented in Figure 14. This approach differs significantly from that

adopted in the current paper. In terms of Catarci et al's [7] classification scheme SPARQLViz is an instance of a form-based VQS; in contrast, NITELIGHT is an instance of a diagram-based system that co-opts ontology browsing and drag-and-drop functionality with a graph-based visualization of query graph patterns. In the absence of any empirical studies it is difficult to comment on the relative merits of these two approaches (i.e. form-based vs. diagram-based); however, comparisons between SPARQLViz and NITELIGHT could (and should) constitute the basis of future experimental studies.

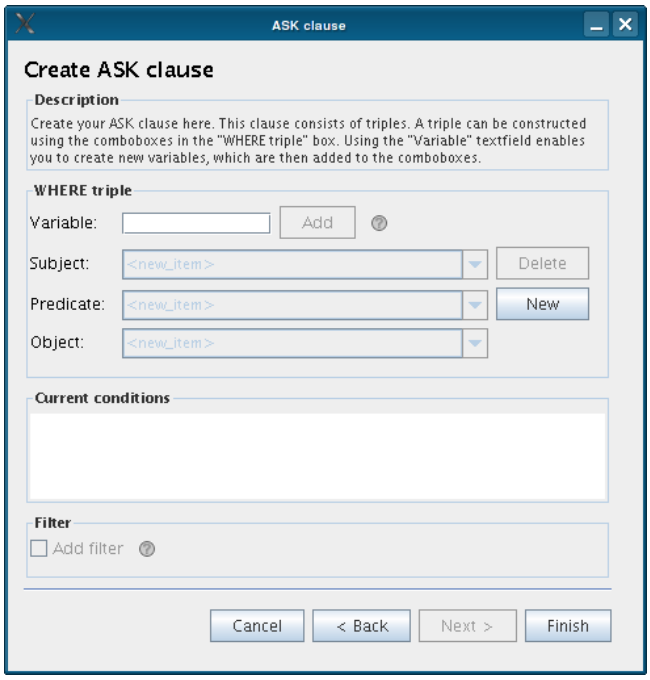

Figure 14. SPARQLViz User Form

One tool that does bear much in common with NITELIGHT is the visual query builder associated with the iSPARQL framework [2] (see Figure 15). The iSPARQL Visual Query Builder supports the user with respect to the specification of all SPARQL query result forms (i.e. SELECT, CONSTRUCT, etc.). It also supports the creation of optional graph patterns as well as UNION combinations of graph patterns in a manner similar to that described for vSPARQL in the present paper. Despite these similarities, differences do exist between the iSPARQL Visual Query Builder and NITELIGHT. Firstly, the visual query language described in this paper (i.e. vSPARQL) is somewhat richer compared to the VQL supported by the iSPARQL Visual Query Builder. vSPARQL supports filter expressions and result ordering as an intrinsic part of its notational syntax, but this information is not available from the set of graphical notations used by iSPARQL (the information is instead provided at the level of editor interface). A second difference concerns the way in which the user is able to access information about target ontologies. The iSPARQL tool relies on a Treeview component that groups ontology elements into 'Concepts' and 'Properties'. NITELIGHT similarly provides access to concepts and properties, but does so using a columnar format that is sensitive to the taxonomic structure of the ontology (see the Ontology Browser section above).

In the absence of empirical studies it is difficult to comment on the significance of the differences between iSPARQL and NITELIGHT in terms of their impact on (e.g.) user approval ratings and query formulation efficiency variables. We would expect the notational differences of the two VQLs to have a relatively minor impact on performance metrics; however, the differences with respect to the tools themselves (e.g. the different ways in which the content of target ontologies is accessed and utilized) may be somewhat more significant. In our experience, understanding the structure of the target ontology as well as the intended meaning of target ontology elements is often the hardest part of the query formulation process.

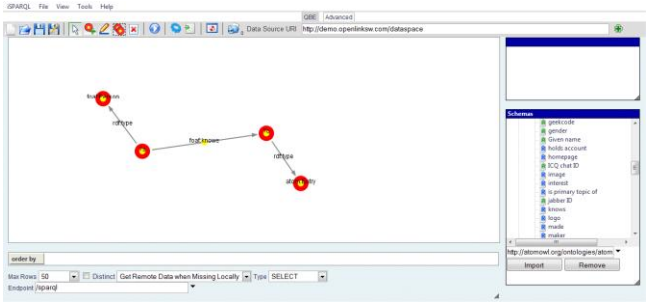

Figure 15: iSPARQL Visual Query Builder

Many of the graphical tools encountered in the literature do not aim to support an underlying text-based language. OntoVQL, for example, does not aim to support query formulation with regard to any specific textual query language (although it does have a partial mapping to nRQL). The tool we describe in this paper does aim to support a specific query language and this motivates a distinction between the current work and some previous studies. We suggest the term Graphical Query Construction System (GQCS) be used to selectively refer to systems that support the visual construction of queries expressed in some other, textual, query language. Systems of this type form a subset of the systems described as VQSs by Catarci et al [7].

Natural Language Query Interfaces

Natural language interfaces provide an alternative to graphical methods of query formulation. These interfaces enable a user to formulate a query using natural language expressions, and they therefore obviate much of the difficulty that novice users may have in terms of creating syntactically valid semantic queries. There have been a number of attempts in the database community to develop systems that use natural language interfaces to support information retrieval [3]. In the context of the Semantic Web, Controlled English interfaces have been used to support information retrieval from semantic repositories [5,

15]. Other systems, such as Aqua-Log [17, 18], provide a question-answering capability that takes queries expressed in natural language and returns answers derived from query execution against a domain ontology. In contrast to the vSPARQL specification described above, natural language interfaces may be more appropriate to users with little or no familiarity with SPARQL.

Semantic Information Browsers

All VQSs aim to support the user with respect to the deliberate creation of queries. The realization of a user's information retrieval goals need not, however, involve the deliberate creation of queries. In some cases, queries can be created and executed (invisibly) as part of an ongoing sequence of goal-directed browsing actions. Systems, such as mSpace [21], for example, support the retrieval of information based on a set of relatively simple and intuitive user interactions, none of which are specifically geared towards query formulation. The question that arises with respect to such systems is whether they undermine the need for tools that explicitly support the query formulation process: couldn't all information retrieval goals be better supported in a system that conflates query generation with episodes of exploratory activity?

While it is certainty true that not every instance of information retrieval necessitates deliberate query formulation, there are, we suggest, cases were users will want to specify information retrieval requests independent of a user interaction context. This is the case when users want to rapidly (re)use the query for information retrieval in multiple contexts, or when they want to distribute a query to other users of a system for the joint evaluation of common result sets. Explicit query design is also required in cases where the query is particularly complex, for example, in cases involving the evaluation of (multiple) variable bindings or disjunctive graph patterns.

FUTURE WORK

The tool described herein was developed as part of an ongoing research program to support human end-users with respect to information retrieval processes in a Semantic Web context. Our future work in this area consists of three activities: extensions to the current tool, development of additional query formulation interfaces and user evaluation studies.

Tool Extensions

The tool described in this paper represents an initial prototype that does not fully support the SPARQL specification. As part of our continued development efforts we aim to extend the functionality of NITELIGHT to include graphical support for all aspects of the SPARQL query language. Of particular interest is the support we aim to provide for the SPARQL CONSTRUCT form. This form of SPARQL query can be viewed as a deductive rule because the query is being used to derive new knowledge from previously asserted facts (see Figure 16). Support for the creation of SPARQL CONSTRUCT queries therefore adds rule editing capabilities to what was originally a tool intended solely for query formulation.

```
CONSTRUCT
{
    ?mine sem:hasType sem:APERS_PI-MI-SR .
    ?mine sem:hasTypeP ".50"^^xsd:float .
}
WHERE
{
     ?incident rdf:type sem:MineIncident .
     ?incident sem:isLocatedIn sem:Afghanistan .
     ?incident sem:involves ?mine .
     ?mine rdf:type sem:Mine .
     ?mine sem:usesExplosive sem:TNT.
}
```

Figure 16. SPARQL CONSTRUCT Query

Another possibility for tool extension relates to use of multiple visual formalisms to represent query elements. As discussed earlier in the paper, Catarci et al [7] present a classification scheme for VQSs that distinguishes between form-based, diagram-based, icon-based and hybrid systems. In its current form, NITELIGHT sits most comfortably in the diagram-based category, although it also includes forms to support query specification and refinement. Subsequent development efforts could, however, extend the range of visual formalisms to include icons (e.g. icons representing types of objects contained in the ontology) and forms (e.g. wizard-like capabilities similar to those described by Borsje and Embregts [6]).

Further extensions and refinements to NITELIGHT include support for creating filter expressions and an ability to update vSPARQL graphical representations based on changes to (text-based) SPARQL queries.

Additional Interfaces

As can be seen from our discussion of related work in this area, there are multiple methods of supporting the human end-user when it comes to query formulation. In addition to the examples presented above (i.e. wizards, QBE systems, graphical designers and natural language interfaces) we can also envision systems providing a range of intellisense, code-completion and syntax checking capabilities, similar to those seen in conventional code-editing environments. One potential direction for future work is therefore to provide a syntax-editing capability that supports expert SPARQL users with respect to the creation and specification of textual queries.

Another type of interface is provided by the use of Controlled English [5, 15] and natural language question-answering systems [17, 18]. These types of systems might be particularly beneficial for novice users who are unfamiliar with semantic query languages. In terms of

extending the capabilities of our current tool with respect to these additional interfaces we aim to develop a natural language query formulation system that implements a similar functionality to that provided by systems such as Aqua-Log [17, 18]. A key difference from the work undertaken with respect to Aqua-Log relates to the serialization of sentential query structures to valid SPARQL queries. At present it is unclear how best to implement this capability. One possibility is to constrain user input using an ontology-specific query grammar; another is to adopt a strategy similar to that seen in the SEWASIE [8] system, wherein the user can progressively select terms in a natural language query and substitute these terms with more specific or general terms based on the domain ontology. Finally, we could opt for a solution based on a subset of natural English, such as Attempto Controlled English (ACE) [10], in which a user is able to express information retrieval requirements using familiar language constructs. In this respect it is interesting to note that ACE can be automatically translated into the N3-style semantic query language PQL [16]. Moreover, a user evaluation of this approach suggests that it promotes the design of good queries with very good retrieval performance [5].

User Evaluation

At this stage we have not performed any user evaluation studies; however, we aim to undertake such studies in the near future. Specific focus areas for evaluation include the general usability of the tool, the ability of the tool to support users with regard to query formulation and comparative analyses of the tool with other graphical [e.g. 8] and non-graphical [e.g. 5] query formulation interfaces. Of particular interest are proposed comparisons between NITELIGHT, SEWASIE [8], SPARQLViz [6], and iSPARQL [2].

Clearly, there are a number of dependent variables that might be assessed in the context of user evaluation studies. These include:

- Syntactic Validity: the number of syntactic errors made during query formulation.
- Query Accuracy: the extent to which the query returns the right information.
- Query Comprehensibility: the level of comprehension attained by a user about a specific query.
- User Satisfaction: subjective ratings of the user's satisfaction with the tool.
- Query Formulation Efficiency: the amount of time taken to formulate queries.

The initial evaluation of NITELIGHT will be based on our target user community (viz., experienced SPARQL users).

CONCLUSION

This paper has presented a graphical editing environment for the construction of semantic queries based on the SPARQL language specification. The tool, called NITELIGHT, is primarily intended for use by those with previous experience of SPARQL (although it could also potentially serve as a support tool for novice users who aim to acquire SPARQL expertise). NITELIGHT is a type of VQS that specifically supports an existing text-based query language; namely SPARQL. In contrast to the recommendations of some commentators [12] we do not propose to develop a simplified query language for end-users; rather we aim to support end-users with respect to the creation of complex queries using supportive user interfaces and user interaction mechanisms. Our tool is one of growing number of VQSs that are being developed to support information retrieval in the context of the Semantic Web.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K.Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

1 http://www.w3.org/2001/11/IsaViz/ - "IsaViz: A Visual Authoring Tool for RDF"

2 http://demo.openlinksw.com/isparql/ - "OpenLink iSPARQL"

3 Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. Natural Language Interfaces to Databases–An Introduction. *Natural Language Engineering 1, 1* (1995), 29-81.

4 Bailey, J., Bry, F., Furche, T., and Schaffert, S., "Web and Semantic Web Query Languages: ASurvey," in *Reasoning Web: First International Summer School*. Msida, Malta, 2005.

5 Bernstein, A., Kaufmann, E., Gohring, A., and Kiefer, C. Querying ontologies: A controlled english interface for end-users. *Proc. 4th International Semantic Web Conference (ISWC05)* (2005), 112–126.

6 Borsje, J., and Embregts, H., "Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface," in *Erasmus School of Economics and Business Economics*, vol. Bachelor. Rotterdam: Erasmus University, 2006.

7 Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing 8, 2* (1997), 215-260.

8 Catarci, T., Dongilli, P., Mascio, T. D., Franconi, E., Santucci, G., and Tessaris, S. An ontology based visual tool for query formulation support. In *16th European Conference on Artificial Intelligence* (2004)

9 Fadhil, A., and Haarslev, V., "OntoVQL: A Graphical Query Language for OWL Ontologies," in *International Workshop on Description Logics (DL-2007)*. Brixen-Bressanone, Italy, 2007.

10 Fuchs, N. E., and Schwitter, R., "Attempto Controlled English (ACE)," in *1st International Workshop on Controlled Language Applications (CLAW 96)* Leuven, Belgium, 1996.

11 Harel, D. On visual formalisms. *Communications of the ACM 31, 5* (1988), 514-530.

12 Hoang, H. H., and Tjoa, A. M., "The virtual query language for information retrieval in the semanticLIFE framework," in *International Workshop on Web Information Systems Modeling*. Trondheim, Norway 2006.

13 Hutt, K., "A Comparison of RDF Query Languages," 2005.

14 Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., and Scholl, M., "RQL: a declarative query language for RDF," in *11th International World Wide Web Conference*. Budapest, Hungary, 2002, pp. 592-603.

15 Kaufmann, E., and Bernstein, A., "How Useful are Natural Language Interfaces to the Semantic Web for Casual End-Users?," in *6th International Symantic Web Conference (ISWC 2007)*. Busan, Korea, 2007.

16 Klein, M., and Bernstein, A. Toward high-precision service retrieval. *Internet Computing, IEEE 8, 1* (2004), 30-36.

17 Lopez, V., and Motta, E. Ontology-driven question answering in AquaLog. In *9th International Conference on Applications of Natural Language to Information Systems* (2004)

18 Lopez, V., Pasin, M., and Motta, E., "AquaLog: An Ontology-portable Question Answering System for the Semantic Web," in *2nd European Semantic Web Conference (ESWC 2005)*. Heraklion, Greece, 2005, pp. 546–562.

19 McBride, B. Jena: a Semantic Web toolkit. *Internet Computing, IEEE 6, 6* (2002), 55-59.

20 http://www.w3.org/2001/sw/DataAccess/rq23/ - "SPARQL Query Language for RDF"

21 schraefel, m. c., Smith, D. A., Owens, A., Russell, A., Harris, C., and Wilson, M. L. The evolving mSpace platform: leveraging the Semantic Web on the trail of the memex. In *Hypertext 2005* (2005)

22 http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/ - "RDQL - A Query Language for RDF"

23 Zloof, M. M. Query-by-Example: A Data Base Language. *IBM Systems Journal 16, 4* (1977), 324-343.

Re-examining the Potential Effectiveness of Interactive Query Expansion

Ian Ruthven
Department of Computer and Information Sciences
University of Strathclyde
Glasgow. G1 1XH
Ian.Ruthven@cis.strath.ac.uk

ABSTRACT

Much attention has been paid to the relative effectiveness of interactive query expansion versus automatic query expansion. Although interactive query expansion has the potential to be an effective means of improving a search, in this paper we show that, on average, human searchers are less likely than systems to make good expansion decisions. To enable good expansion decisions, searchers must have adequate instructions on how to use interactive query expansion functionalities. We show that simple instructions on using interactive query expansion do not necessarily help searchers make good expansion decisions and discuss difficulties found in making query expansion decisions.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: - search process, relevance feedback.

General Terms

Experimentation, Human Factors

Keywords

Query expansion, Evaluation

1. INTRODUCTION

Query expansion techniques, e.g. [1, 5], aim to improve a user's search by adding new query terms to an existing query. A standard method of performing query expansion is to use relevance information from the user – those documents a user has assessed as containing relevant information. The content of these relevant documents can be used to form a set of possible expansion terms, ranked by some measure that describes how useful the terms might be in attracting more relevant documents, [13]. All or some of these expansion terms can be added to the query either by the user – *interactive query expansion* (**IQE**) – or by the retrieval system – *automatic query expansion* (**AQE**).

SIGIR'03, July 28–August 1, 2003, Toronto, Canada.

One argument in favour of AQE is that the system has access to more statistical information on the relative utility of expansion terms and can make better a better selection of which terms to add to the user's query. The main argument in favour of IQE is that interactive query expansion gives more *control* to the user. As it is the user who decides the criteria for relevance in a search, then the user should be able to make better decisions on which terms are likely to be useful, [10].

A number of comparative user studies of automatic versus interactive query expansion have come up with inconclusive findings regarding the relative merits of AQE versus IQE. For example, Koenemann and Belkin [10] demonstrated that IQE can outperform AQE for specific tasks, whereas Beaulieu [1] showed AQE as giving higher retrieval effectiveness in an operational environment. One reason for this discrepancy in findings is that the design of the interface, search tasks and experimental methodology can affect the uptake and effectiveness of query expansion techniques.

Magennis and Van Rijsbergen [12] attempted to gauge the effectiveness of IQE in live and simulated user experiments. In their experiments they estimated the performance that might be gained if a user was making very good IQE decisions (the *potential* effectiveness of IQE) compared to that of real users making the query modification decisions (the *actual* effectiveness of IQE). Their conclusion was that users tend to make sub-optimal decisions on query term utility.

In this paper we revisit this claim to investigate more fully the potential effectiveness of IQE. In particular we investigate how good a user's query term selection would have to be to increase retrieval effectiveness over automatic strategies for query expansion. We also compare human assessment of expansion term utility with those assessments made by the system.

The remainder of the paper is structured as follows. In section 2 we discuss the motivation behind our investigation and that of Magennis and Van Rijsbergen. In section 3 we describe our experimental methodology and data. In section 4 we investigate the potential effectiveness of IQE and in section 5 we compare potential strategies for helping users make IQE decisions. In section 6 we summarise our findings.

2. MOTIVATION

In this section we summarise the experiments carried out by Magennis and Van Rijsbergen, section 2.1, some limitations of these experiments, section 2.2, and discuss the motivation behind our work, section 2.3

2.1 The potential effectiveness of IQE

In [11, 12] Magennis and Van Rijsbergen, based on earlier work by Harman [7], carried out an experiment to estimate how good IQE could be if performed by expert searchers.

Using the WSJ (1987-1992) test collection, a list of the top 20 expansion terms was created for each query, using terms taken from the top 20 retrieved documents. This list of possible expansion terms was ranked by applying the F4, [14], term reweighting formula to the set of *unretrieved* relevant documents. This set of documents consists of the relevant documents *not* yet seen by the user. This set could not be calculated in a real search environment as retrieval systems will only have knowledge of the set of documents that *have* been seen by the user. However, as query expansion aims to retrieve this set of documents, they form the best evidence on the utility of expansion terms.

Using these sets of expansion terms, Magennis and Van Rijsbergen simulated a user selecting expansion terms over four iterations of query expansion. At each iteration, from the list of 20 expansion terms, the top 0, 3, 6, 10 and 20 terms were isolated. These groups of terms simulated possible sets of expansion terms chosen by a user. By varying which group of terms was added at each iteration, all possible expansion decisions were simulated. For example, expansion by the top 3 terms at feedback iteration 1, the top 10 terms at feedback iteration 2, etc. The best simulation for each query was taken to be a measure of the best IQE decisions that could be made by a user; the *potential* effectiveness of IQE.

2.2 Limitations

One of the benefits of an approach such as that taken by Magennis and Van Rijsbergen is that it is possible to isolate the effect of query expansion itself. That is, by eliminating the effects of individual searchers and search interfaces the results can be used as baseline figures with which to compare user search effectiveness. However, in [11, 12], Magennis noted several limitations of this particular measurement of the potential effectiveness of IQE.

i. only certain combinations of terms are considered, i.e. the top 3, 6, 10 or 20 terms. Other combinations of terms are possible, e.g. the top 4 terms, and these may give better retrieval performance.

ii. real searchers are unlikely to use add a *consecutive* set of expansion terms, i.e. the top 3, 6, 10 or 20 terms suggested by the system. It is more likely that searchers will choose terms from throughout the list of expansion terms. In this way, users can, for example, avoid poor expansion terms suggested by the system.

iii. the ranking of expansion terms is based on information from the *unseen* relevant documents; ones that the user has not yet viewed. In a real search environment the expansion terms will be ranked based on their presence or absence in the documents seen and assessed relevant by the user.

iv. only one document collection was used. Differences in the creation of test collections, the search topics used and the documents present in the test collection may affect the results of their conclusions and restrict the generality of their conclusions.

2.3 Aims of study

In our experiments we aim to overcome these limitations to create a more realistic evaluation of the potential effectiveness of interactive query expansion. In particular we aim to investigate how good IQE could be, how easy it is to make good IQE decisions and investigate guidelines for helping users make good IQE decisions. We also investigate what kind of IQE decisions are actually made by searchers when selecting new search terms. In the following section we describe how we obtain the query expansion results analysed in the first part of this paper. These experiments are also based on *simulations* of interactive query expansion decisions.

3. EXPERIMENTAL SETUP

In this section we outline the experimental methodology we used to simulate query expansion decisions. The experiments themselves were carried out on the Associated Press (**AP** 1998), San Jose Mercury News (**SJM** 1991), and Wall Street Journal (**WSJ** 1990-1992) collections, details of which are given in Table 1. These collections come from the TREC initiative [16].

Table 1: Collection statistics

	AP	SJM	WSJ
Number of documents	79919	90257	74520
Number of queries used	32	39	28
Average words per query[1]	2.9	3.7	2.9
Average number of relevant documents per query	37.8	58.6	30.3

For each query we use the top 25 retrieved documents to provide a list of possible expansion terms, as described below. Although each collection comes with a list of 50 topic (query) descriptions, we concentrate on those queries where query expansion *could* change the effectiveness of an existing query. This meant excluding some queries from each test collection; those queries for which there are no relevant documents, queries where *no* relevant documents were retrieved in the top 25 documents (as no expansion terms could be formed without at least one relevant document), and queries where *all* the relevant documents are found within the top 25 retrieved documents (as query expansion will not cause a change in retrieval effectiveness for these queries).

In our experiments we used the *wpq* method of ranking terms for query expansion, [13], as this has been shown to give good results for both AQE and IQE, [4]. The equation for calculating a weight for a term using *wpq* is shown below,

[1]Queries used in the experiments only. The query comes from the short *title* field of the TREC topic description.

where the value r_t = the number of seen relevant documents containing term *t,* n_t = the number of documents containing *t*, *R* = the number of seen relevant documents for query *q, N* = the number of documents in the collection.

$$wpq_t = \log \frac{r_t/(R-r_t)}{(n_t-r_t)/(N-n_t-R+r_t)} \bullet \left(\frac{r_t}{R} - \frac{n_t-r_t}{N-R}\right)$$

Our procedure was as follows:

For each query,

i. rank the documents using a standard *tf*idf* weighting to obtain an initial ranking of the documents.

ii. use the relevant documents in the top 25 retrieved documents to obtain a list of possible expansion terms, using the *wpq* formula to rank the expansion terms.

iii. using the top 15 expansion terms, create all possible sets of expansion terms. For each query this gives 32 678 possible sets of expansion terms. This simulates all possible selections of expansion terms using the top 15 terms, including no expansion of the query. Each of these 32 678 sets of terms represents a *possible* IQE decision that could be made by a user.

iv. using each combination of expansion terms, add the combination to the original query and use the new query to rank the documents, again using *tf*idf*. That is, for each query, we carry out 32 678 separate *versions* of query expansion.

v. calculate the recall-precision values for each version of the query. Here we use a full-freezing approach by which we only re-rank the unseen documents – those not use to create the list of expansion terms. This is a standard method of assessing the performance of a query expansion technique based on relevance information, [3]

We only use the top 15 expansion terms for query expansion as this is a computationally intensive method of creating possible queries. In a real interactive situation users may be shown more terms than this. However, it does allow us to concentrate on those terms that are considered by the system to be the best for query expansion.

For each query in each collection, therefore, we have a set of 32 678 possible IQE decisions that could be made by a searcher. For each possible IQE decision we can assess the effect of making this decision on the quality of the expanded query. We use this information in several ways; firstly, in section 4, we compare the possible IQE decisions against three methods of applying AQE. We then, in section 5, examine potential strategies for helping searchers make good IQE decisions. In section 5 we also compare the possible IQE decisions against human expansion decisions.

4. COMPARING QUERY EXPANSION TECHNIQUES

In this section we examine the potential effectiveness of IQE against three possible strategies for applying AQE. In this section we compare how likely a user is to make better query expansion decisions using IQE than allowing the system to perform AQE. Our three AQE techniques are:

Collection independent expansion. A common approach to AQE is to add a fixed number of terms, *n*, to each query. Our first AQE technique simulates this by adding the top six expansion terms to all queries, irrespective of the collection used. The value of six was chosen without prior knowledge of the effectiveness of adding this number of terms to any of the queries in the test collections used.

Collection dependent expansion. The previous approach to AQE adds the same number of expansion terms to all queries in all collections. When using a specific test collection we can calculate a better value of *n*; one that is specific to the test collection used. To calculate *n*, for each collection, we compared the average precision over all the queries used in each collection after the addition of the top *n* expansion terms, where *n* varied from 1 to 15. The value of *n* that gave the optimal value of average precision for the whole query set was taken to be the value of *n* for each query in the collection.

These values could not be calculated in an operational environment, where knowledge of all queries submitted is unknown. However, it gives a stricter AQE baseline measure as the value of *n* is optimal for the collection used. The values for *n* are shown in Table 2, and is higher than the six terms added in the previous strategy.

Table 2: Optimal values of *n*

Collection	AP	SJM	WSJ
n	15	15	13

Query dependent expansion. The collection dependent expansion strategy adds a fixed number of terms to each query within a test collection. This is optimal for the entire query set but may be sub-optimal for individual queries, i.e. some queries may give better retrieval effectiveness for greater or smaller values of *n*. The query dependent expansion strategy calculates which value of *n* is optimal for individual queries. This may be implemented in an operational retrieval system by, for example, setting a threshold on the expansion term weights.

These three AQE methods act as baseline performance measures for comparing AQE with IQE.

4.1 Query expansion vs. no query expansion

We first compare the effect of query expansion against no query expansion; how good are different approaches to query expansion? In Table 3 we compare the AQE baselines against no query expansion: **t**he performance of the original query with no additional query terms. Specifically, we compare how many queries in each collection give higher average precision than no query expansion; the *percentage* of queries that are improved by each AQE strategy. Also included in this table, in bold figures, are the average precision figures given by applying the techniques.

As can be seen, all AQE strategies were more likely, on average, to improve a query than harm it. That is, all techniques

improved at least 50% of the queries where query expansion could make a difference to retrieval effectiveness.

The automatic strategy that is most *specific* to the query, the query dependent strategy, not only improves the highest percentage of queries– is most *stable* – but also gives the highest average precision over the queries - is most *effective*. Conversely the automatic strategy that is least effective and improves least queries is the one that is less tailored to either the query or collection – the collection independent strategy.

Table 3: AQE baselines and example IQE decisions.

Baseline	AP	SJM	WSJ
Collection independent	56%	72%	50%
	18.8	**23.8**	**18.4**
Collection dependent	72%	79%	53%
	19.0	**24.8**	**18.6**
Query dependent	75%	90%	86%
	20.1	**26.7**	**21.1**
IQE best	94%	97%	96%
	22.3	**29.1**	**22.4**
IQE middle	31%	38%	30%
	18.4	**22.9**	**18.1**
IQE worst	0%	0%	0%
	11.9	**15.8**	**14.0**

We can compare these decisions against possible IQE decisions. Firstly, in row 5 of Table 3, we show the percentage of queries improved, and average precision obtained, when using the *best* IQE decision for each query. This set of figures gives the best possible results on each collection when using query expansion. This is the highest potential performance of IQE using the top 15 expansion terms.

Comparing the performance of the best IQE decision against the AQE decisions, it can be seen that IQE has the potential to be the most *stable* technique overall in that it improves most queries. It also has the potential to be the most effective query expansion technique as it gives highest overall average precision. However this is only a *potential* benefit, as we shall show in the remainder of this paper it may not be easy for a user to select such an optimal set of terms.

For example, in the row 6 of Table 3 we show the performance of a middle-performing IQE decision. This is obtained, for each query, by ranking the average precision of all 32768 possible IQE decisions and selecting the IQE decision at position 16384 (half way down the ranking). This decision is one that would be obtained if a user makes query expansion decisions that were neither good nor poor compared to other possible decisions. This result shows that even fair IQE decisions can perform relatively poorly; improving less than half of queries and giving poorer retrieval effectiveness than any of the AQE strategies.

Finally, in row 7 of Table 3, we show the effect if a user was consistently making the worst IQE decisions possible, i.e. always choosing the combination of expansion terms that gave the lowest average precision of all possible decisions. Even though a user is unlikely to *always* make such poor decisions, these decisions are being made on terms selected from the top 15 expansion terms. So, although IQE *can* be effective it is a technique that needs to be applied carefully. In the next section we examine how likely a user is to make a *good* decision using IQE.

4.2 AQE vs. IQE

In this section we look at how difficult it is to select a set of expansion terms that will perform better than AQE or no query expansion. We do this by comparing how many of the possible IQE decisions will give better average precision than the AQE baselines. In Table 4 we show the results of this analysis. For each collection we show how many possible IQE decisions gave greater average precision than each of the three baselines (top row in columns 2-4) and how many of the decisions gave a *significantly* higher average precision than the baselines (bold figures in columns 2 – 4)[2].

Table 4: Percentage of combinations better than baselines

Baseline	AP	SJM	WSJ
No expansion	59%	69%	53%
	30%	**38%**	**21%**
Collection independent	45%	36%	41%
	9%	**11%**	**12%**
Collection dependent	47%	35%	43%
	13%	**9%**	**8%**
Query dependent	9%	10%	10%
	1%	**1%**	**2%**

What we are trying to uncover here is how likely a user is to make good IQE decisions over a range of queries. The argument for IQE, based on this analysis, is mixed. On the positive side over 50% of the *possible* IQE decisions give better performance than no query expansion, and over 20% of the possible decisions give significantly better performance (row 2). However, this also means that nearly half of the possible decisions will decrease retrieval performance[3] and most decisions will not make any significant difference to the existing query performance.

Compared against the best AQE strategy (query dependent), only a small percentage (9-10%) of possible decisions are likely to be better than allowing the system to make the query expansion decisions. Based on this analysis it

[2] Measured using a *t*-test ($p < 0.05$), holding recall fixed and varying precision. Values were calculated on the set of RP figures for each query not the averaged value.

[3] A small percentage (1%-3%) of possible decisions will neither increase nor decrease query performance.

appears that it may be hard for users to make very good IQE decisions; ones that are better than a good AQE technique.

The collection independent strategy is the most realistic default AQE approach as it assumes no knowledge of collections or queries. However, although 35%-45% of possible IQE decisions are better than the collection independent strategy, this still means that searchers are more likely to make a poorer query expansion decision than the system. This is only true, however, if users lack any method of selecting good combinations of expansion terms. In the next section we analyse potential guidelines that could be given to users to help them make good IQE decisions.

5. POSSIBLE GUIDELINES FOR IQE

In this section we try to assess possible instructions that could be given to users to help them make use of IQE as a general search technique.

5.1 Select more terms

One reason for asking users to engage in IQE is to give more evidence to the retrieval system regarding the information for which they are looking. Users, especially in web searches, often use very short queries [9]. Presenting lists of possible expansion terms is one way to get users to give more information, in the form of query words, to the system.

A useful guideline to give to users, then, may be to expand the query with as many useful terms are possible. In Table 5 we compare the size of IQE decisions that lead to an increase in retrieval effectiveness (*good* IQE decisions, Table 5, row 4) against those that led to a decrease in retrieval effectiveness (*poor* IQE decisions, Table 5, row 5). As can be seen, the size of the query expansion does not distinguish good decisions from poor decisions.

The size of the *best* IQE decisions (the average size of the combinations that gave the best average precision) is similar both to the average size of the good and poor combinations (Table 5, row 3). The sizes of the average of the best AQE decisions are also within a similar range (Table 5, row 2). So giving the system *more* evidence does not necessarily gain any improvement in effectiveness.

Table 5: Average size of query expansions

	AP	SJM	WSJ
Query dependent	6.63	7.10	8.46
IQE best	7.29	5.56	7.16
IQE good	7.35	7.60	7.27
IQE poor	7.61	7.27	7.44

5.2 Trust the system

A second approach might be to advise users to concentrate on the terms suggested most strongly by the system. These are terms that are calculated by the system to be the most likely to improve a query, and in our experiment are the terms with the highest *wpq* score. In Table 6, we present the average *wpq* value of the terms chosen in good and poor IQE decisions, and also in the best IQE and AQE strategies.

The average *wpq* value for terms in good (row 4) and poor IQE decisions (row 5) is relatively similar. This means that sets of terms with high *wpq* values are not more likely to give good performance than sets of terms with lower *wpq* values.

The average value for the best AQE decisions (row 2) is generally higher than that of the IQE decisions. This, however, results in part from the fact that the query dependent AQE strategy adds a consecutive set of terms taken from the top of the expansion term ranking. As these terms are at the top of the term ranking, they will naturally have a higher *wpq* value.

The average term score for the *best* IQE (row 3) decision is also higher than either the good or poor IQE decisions, so there is some merit in choosing terms that the system recommends most highly – those with high *wpq* values.

Table 6: Average *wpq* of terms chosen

	AP	SJM	WSJ
Query dependent	2.20	2.11	2.39
IQE best	2.94	1.91	2.26
IQE good	1.92	1.71	1.70
IQE poor	1.93	1.70	2.12

However, the lack of difference between the good and poor IQE decisions means we cannot *alone* recommend the user concentrates more closely on the terms suggested by the system. That is, highly scored terms are useful but the user must apply some additional strategy to select which of these terms to use for query expansion.

5.3 Use semantics

One of the more intuitive arguments in favour of IQE is that, unlike the statistically-based query expansion techniques, humans can exploit semantic relationships for retrieval. That is, people can recognise expansion terms that are semantically related to the information for which they are seeking and expand the query using these terms. However, investigations such as the one presented in [2] indicate that searchers can find it difficult to use semantic information even when the system supports the recognition and use of semantic relationships.

Consequently, in this section we outline a small pilot experiment designed to compare system recommendations of term utility against human assessment of the same terms.

5.3.1 System analysis of expansion term utility

The system, or *automatic*, analysis of an expansion term is based on the overall impact of adding that term to all possible IQE decisions that do not already contain the term. That is, we estimate the *likely* impact of adding a new expansion term *t* to an existing set of expansion terms.

For each query, each expansion term, *t*, belongs to 50% (16384) of the possible IQE decisions (and does not belong to 50% possible decisions, including no query expansion). In

effect these two sets of possible decisions are identical except as relates to *t*: adding *t* to each IQE decision in the latter set would give an IQE decision in the former set. By comparing the average precision of all IQE decisions that contain *t*, with the corresponding decisions that do not contain *t*, we can classify each of the top 15 expansion terms according to whether they are *good*, *neutral* or *poor* expansion terms. Good terms are those that are likely to improve the performance of a possible IQE decision (a set of expansion terms); neutral ones are those that generally make no difference and poor expansion terms are those that are likely to decrease the performance of a set of expansion terms.

We demonstrate this in Table 7, based on the TREC topic 259 '*New Kennedy Assassination Theories*' run on the AP collection. Each row shows what percentage of the 16384 possible decisions, not already containing the term in column 1, that are improved, worsened, or have no difference *after* the addition of the term. For example, the addition of the term *jfk* will always improve retrieval effectiveness. That is, adding the term *jfk* to any set of expansion terms will increase retrieval effectiveness. Conversely, adding the term *frenchi* will always reduce the retrieval effectiveness, and the addition of the term *warren*[4] will make no difference.

Table 7: Addition of expansion terms for TREC topic 259

Term	Improved	No difference	Worsened
jfk	100	0	0
oswald	3	0	97
dealei	37	4	59
kwitni	29	0	71
motorcad	64	4	32
marcello	100	0	0
warren	0	100	0
theorist	0	100	0
theori	18	0	82
depositori	67	0	33
documentari	40	19	41
belin	0	100	0
tippit	46	8	46
frenchi	0	0	100
bulletin	45	0	55

For simplicity, we classify terms simply by their predominant tendency. For the example in Table 7 the good terms are *jfk*, *motorcad*, *marcello*, and *depositori*. The poor terms are *oswald*, *dealei*, *kwitni*, *theori*, *documentari*, *frenchi* and *bulletin*, and the neutral terms are *warren*, *theorist* and *belin*. The term *tippit* is good and poor for an equal percentage of combinations and cannot be classified.

5.3.2 Human analysis of expansion term utility

The automatic classification of expansion term utility presented in the previous section was compared against a set of human classification of the same expansion terms.

We selected 8 queries from each collection and asked 3 human subjects to read the whole TREC topic and each of the relevant documents found within the top 25 retrieved documents. These were the relevant documents used to create the list of the top 15 expansion terms in the previous experiment. The subjects were given the full TREC topic description to provide some context for the search, and were shown the initial query that retrieved the documents. The subjects were then presented with the top 15 expansion terms. For each expansion term the subjects were asked whether they felt the term would be useful or not useful at retrieving additional relevant documents when added to the existing query[5].

We asked each subject to assess each of the 24 queries rather than distributing the queries across multiple subjects. This was to preserve any strategies the individual users may be employing when selecting expansion terms [8]. However, we did not ask the subjects to read the *non*-relevant retrieved documents as we felt this was too great a burden on the subjects.

The subjects' selection of expansion terms was compared against the automatic analysis from section 5.3.1 to compare the system classification against human classification of expansion term utility. The comparison was done in three ways; first we compare how good the subjects are at detecting good expansion terms, section 5.3.2.1, how good the subjects are at eliminating poor expansion terms, section 5.3.2.2, and examine the decisions made by the subjects, section 5.3.2.3.

5.3.2.1 Detecting good expansion terms

For each subject we examine first whether the subjects can detect *good* expansion terms; whether the subjects can recognise the expansion terms that are likely to be useful in combination with other expansion terms.

Table 8: Percentage of good expansion terms detected

	Subject 1	Subject 2	Subject 3
AP	73%	60%	63%
SJM	50%	40%	42%
WSJ	62%	32%	45%

In Table 8 we show the percentage of the good expansion terms, as classified in section 5.3.1, which were chosen by each

[4] From the Warren Commission which investigated the assassination of President Kennedy. This term and the term *theori* are the only ones to appear in the TREC topic description.

[5] If the subjects could not decide whether the term was useful/not useful, they could assign the term to the category 'cannot decide'.

subject as being possibly useful for query expansion. The subjects varied in their ability to identify good expansion terms, being able to identify 32% - 73% of the good expansion terms.

5.3.2.2 Eliminating poor expansion terms

If the subjects are not always good at detecting good expansion terms perhaps they are better are eliminating poor expansion terms? In Table 9 we show the percentage of expansion terms that were assessed as being poor by the system but good by the subjects. As in the previous section, the subjects' ability to correctly classify expansion terms varied with at least 25% of the poor expansion terms being rated as good by the subjects. The implication here is that subjects may have difficulty spotting poor expansion terms.

Table 9: Percentage of poor expansion terms classified as good by subjects

	Subject 1	Subject S2	Subject S3
AP	54%	36%	43%
SJM	39%	26%	35%
WSJ	38%	45%	39%

One reason for the poor classification of terms may be that the subjects are only choosing certain types of terms. In Table 10 we compare the cases where the system classification (column 2) agreed or disagreed with the subjects' classification (column 3) of terms.

Table 10: Comparison of system and subject classification

	System	User	S1	S2	S3
AP	**Good**	**Good**	692 (6.5)	570 (6.22)	666 (5.3)
	Poor	**Good**	622 (4.3)	914 (4.48)	601 (4.2)
	Good	**Poor**	429 (4.5)	830 (4.14)	578 (4.1)
	Poor	**Poor**	142 (1.7)	223(1.8)	178 (1.6)
SJM	**Good**	**Good**	1321 (7.5)	1831 (7.3)	1542 (7.7)
	Poor	**Good**	766 (3.7)	867 (3.7)	802 (3.7)
	Good	**Poor**	390 (2.8)	405 (3.8)	397 (3.5)
	Poor	**Poor**	53 (1.4)	253 (1.9)	179 (1.6)
WSJ	**Good**	**Good**	833 (5.2)	204 (2.2)	765 (4.5)
	Poor	**Good**	1496 (3.9)	682 (2.8)	881 (3.3)
	Good	**Poor**	285 (2.6)	598 (4.0)	270 (2.7)
	Poor	**Poor**	427 (1.8)	966 (3.0)	470 (2.3)

For each case we give the average collection occurrence of the terms and (the figure in parentheses) their average occurrence within the relevant documents. For example, for the terms on which subject 1 and the system agreed that the terms were useful, these terms appeared in an average of 692 documents in the AP collection and an average of 6.5 relevant documents.

Appearing in lots of relevant documents appears initially to correlate with an assessment of good expansion term utility. However the difference in relevant document occurrence between good/poor and bad/poor misclassification is often slight.

The most apparent pattern from Table 10 is that subjects tend to classify terms with a high *collection* frequency as being good expansion terms. Conversely terms with a low collection frequency are likely to be assessed as being poor expansion terms. This is not a universal pattern (Subject 2 on the WSJ collection for example does the opposite) but it is the main pattern and suggests that searchers may not being assessing which terms are useful but which terms are *recognisable*.

5.3.3 Subjects' reasons for expansion term selection

We discussed with each subject their reasons for their classification of expansion terms. Based on the subjects' reasons for classification and the later automatic classification, we can suggest three reasons for misclassification of expansion term utility.

i. *Statistical relationships are important as well as semantic ones*. Subjects tended to ignore terms if the terms appear to have been suggested for purely statistical reasons, e.g. numbers. In general this may be a sensible approach if the query does not mention specific numbers or dates. However, the documents in the static collections we used are only a sample of the *possible* documents on the topics investigated. In this case, strong statistical relationships may be useful for future retrieval.

ii. *Users cannot always identify semantic relationships*. Making good use of semantic information means being able to identify semantic relationships between the information need and the possible expansion terms. For specialised or unusual terms, the subjects could be unsure of the value of these terms unless the relationship between these terms and the information need was made clear in the documents.

However, being able to recognise why expansion terms have been suggested, and the searcher's ability to classify terms as useful or not, does not necessarily guarantee that the terms themselves will be seen as useful. Rather, we propose that searchers need more sophisticated support in assessing the potential quality of expansion terms.

iii. *Users cannot always identify useful semantic relationships for retrieval*. The difficulty most subjects experienced with selecting expansion terms is that, although they felt they could identify obvious semantic relationships, they could not identify which semantic relationships were going to attract more relevant documents. In short, the subjects felt they could not identify the effect of individual expansion terms on future retrieval. Instead the subjects concentrated mainly on terms they viewed as safe; those that were semantically related to the *topic* description rather than the retrieved relevant documents. That is, the subjects tended to concentrate on terms for *new* queries rather than modified or refined queries.

This type of decision-making can also be seen in other investigations, e.g. [15] which demonstrated that, although

terms suggested from relevant documents *can* be useful terms, they are often not used as a main source of additional search terms.

In a real interactive environment users can, of course, try out expansion terms, or add their own new terms, and see the effect on the type of documents retrieved. However, the lack of connection between expansion terms and documents used to provide those terms indicates that searchers may need more support in how to use query expansion as a general interactive technique.

6. CONCLUSIONS

In this paper we examined the potential effectiveness of interactive query expansion. This is mainly a simulation experiment and is intended to supplement rather than replace experimental investigations of real user IQE decision-making. There are several limitations to this work: for example, we only concentrated on altering the content of the query; future investigations will compare the results obtained here when we use relevance weighting in addition to query expansion. We also do not differentiate between queries although the success of query expansion can vary greatly across queries. We will consider this in future work, our intention here is to investigate the *general* applicability of query expansion.

The experimental results initially provided a comparison between AQE and IQE techniques. From Table 3, section 4.1, IQE has the potential to be an effective technique compared with AQE. One of the main claims for IQE is that searchers can be more adept, than the system, at identifying good expansion terms. This may be particularly true for certain types of search, e.g. in [6] Fowkes and Beaulieu showed that searchers preferred IQE when dealing with complex query statements. Subjects may also be better at targeting specific aspects of the search, i.e. focussing on parts of their information need.

However, the analyses presented here show that the potential benefits of IQE may not be easy to achieve. In particular searchers have difficulty identifying useful terms for effective query expansion. The implication is that simple term presentation interfaces are not sufficient in providing sufficient support and context to allow *good* query expansion decisions. Interfaces must support the identification of relationships between relevant material and suggested expansion terms and should support the development of good expansion strategies by the searcher.

7. REFERENCES

[1] Beaulieu, M. Experiments with interfaces to support query expansion. Journal of Documentation. 53. 1. pp 8-19. 1997.

[2] Blocks, D., Binding, C., Cunliffe, D., and Tudhope, D**.** Qualitative evaluation of thesaurus-based retrieval**.** Proceedings of the 6th European Conference in Digital Libraries. Rome. Lecture Notes in Computer Science 2458. pp 346-361. 2002.

[3] Chang, Y. K., Cirillo, C., and Razon, J. Evaluation of feedback retrieval using modified freezing, residual collection & test and control groups. The SMART retrieval system - experiments in automatic document processing. G. Salton (ed). Chapter 17. pp 355-370. 1971.

[4] Efthimiadis, E. N. User-choices: a new yardstick for the evaluation of ranking algorithms for interactive query expansion. Information processing and management. 31. 4. pp 605-620. 1995.

[5] Efthimiadis, E. N.. Query expansion. ARIST Volume 31: Annual Review of Information Science and Technology. Martha E. Williams (ed). 1996.

[6] Fowkes, H., and Beaulieu, M. Interactive searching behaviour: Okapi experiment for TREC-8. Proceedings of IRSG 2000. 22nd Annual Colloquium on Information Retrieval Research. Cambridge. Cambridge. 2002.

[7] Harman, D. Towards interactive query expansion. Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp 321-331. Grenoble. 1988.

[8] Iivonen, M. Consistency in the selection of search concepts and search terms. Information Processing and Management. 31. 2. pp 180-186. 1995.

[9] Jansen, B. J., Spink, A., and Saracevic, T. Real life, real users, and real needs: A study and analysis of users on the web. Information Processing & Management. 36. 2. pp 207-227. 2000.

[10] Koenemann, J., and Belkin, N. J. A case for interaction: a study of interactive information retrieval behavior and effectiveness. Proceedings of the Human Factors in Computing Systems Conference (CHI'96). pp 205-212. Zurich. 1996.

[11] Magennis, M. The potential and actual effectiveness of interactive query expansion. PhD thesis. University of Glasgow. 1997.

[12] Magennis, M., and van Rijsbergen, C. J. The potential and actual effectiveness of interactive query expansion. Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp 324-332. Philadelphia. 1997.

[13] Robertson, S. E. On term selection for query expansion. Journal of Documentation. **46**. 4. pp 359-364. 1990.

[14] Robertson, S. E., and Sparck Jones, K. Relevance weighting of search terms. Journal of the American Society for Information Science. **27**. 3. pp 129-146. 1976.

[15] Spink, A. and Saracevic, T. Interaction in information retrieval: Selection and effectiveness of search terms. Journal of the American Society for Information Science. 48**.** 8. pp 741-761. 1997.

[16] Voorhees, E. H., and Harman, D. Overview of the sixth text retrieval conference (TREC-6). Information Processing and Management. 36. 1. pp 3 - 35. 2000.

Towards Interactive Query Expansion

Donna Harman*

Lister Hill National Center for Biomedical Communications
National Library of Medicine
Bethesda, Maryland, 20209

Abstract

In an era of online retrieval, it is appropriate to offer guidance to users wishing to improve their initial queries. One form of such guidance could be short lists of suggested terms gathered from feedback, nearest neighbors, and term variants of original query terms. To verify this approach, a series of experiments were run using the Cranfield test collection to discover techniques to select terms for these lists that would be effective for further retrieval. The results show that significant improvement can be expected from this approach to query expansion.

1. Introduction

Statistically-based keyword retrieval systems are known for almost always retrieving some documents relevant to a query, but seldom retrieving all documents relevant to that query, at least by a rank most users will see. Many efforts have been made to improve statistically-based keyword performance, but only the use of various types of term weighting for ranking [SPARCK-JONES72, CROFT83, SALTON83, HARMAN86], have made enough improvement to be universally accepted as the state-of-the-art for measuring further improvements. Usually improvements in recall are achieved only at the expense of precision.

One method for retrieving more relevant documents is to expand the query terms by using relevance feedback, conflating word stems, and/or adding synonyms from a thesaurus. These additional terms allow the query to match documents that contain words which are related to the query, but not actually expressed in it. This paper is based on the following premise: a statistically-based keyword system that retrieves a few relevant documents in answer to a query should be able to help the user modify the query in order to retrieve more relevant documents.

As an example, suppose that a user inputs a query, finds several relevant documents in the first screenful of documents shown to them, and then desires to see more documents. The following sample screen might then appear.

*Current address: National Bureau of Standards, Gaithersburg, Maryland, 20899

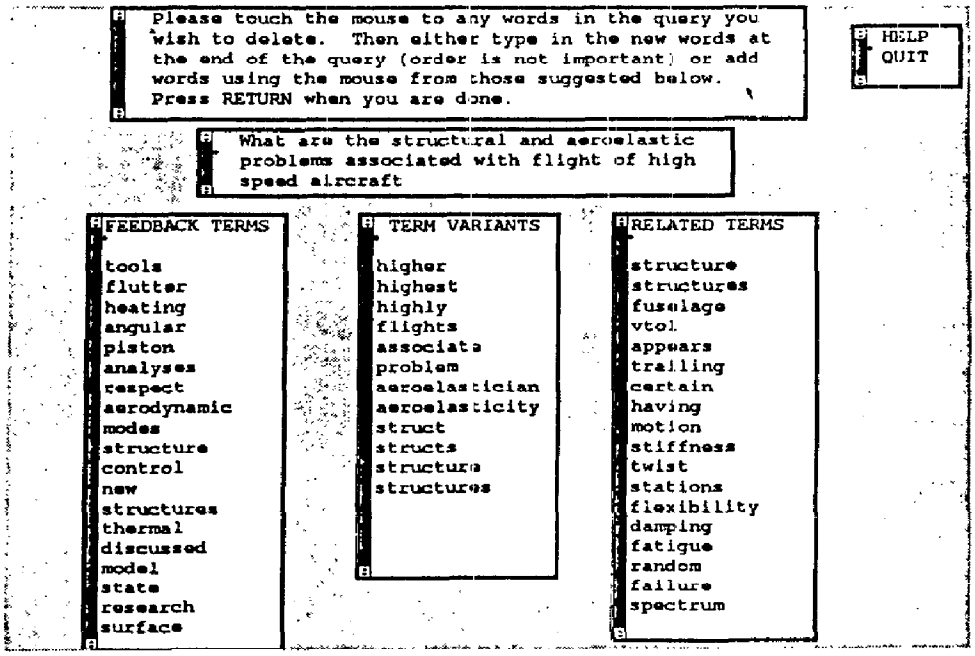

Figure 1 -- Sample Screen for Query Modification

The first window contains words derived from relevance feedback (see below). The second window contains term variants of the original query terms, and the third window contains words derived from a manual thesaurus, or some type of automatic generation of word relations. The size of the windows and the patience of the user require that only a reasonable number of terms be displayed. The goal of this experiment was the discovery of techniques for filling those windows with the terms most likely to increase performance.

2. The use of relevance feedback for query expansion (Window 1)

2.1 Past work

Relevance feedback is an interactive retrieval tool, but can be used in conjunction with a test collection by simulating the user interaction. That is, assuming that the user is shown the first ten documents retrieved, the known relevant documents in that set become the relevant documents used for feedback.

One of the first trials of relevance feedback was by Rocchio in 1965 [ROCCHIO68], where he simulated relevance feedback by using a test collection with known relevant documents. Using the SMART vector approach, Rocchio modified the queries by adding the term vectors for all relevant documents retrieved by a given cutoff, and subtracting the term vectors for all non-relevant documents retrieved by that cutoff. This effectively added many terms, some with negative weights; increased the weights of some query terms; and decreased the weights of others. The results, however, were not conclusive because of problems in evaluating relevance feedback in general.

Wu [WU81] also expanded the queries using the SMART vector expansion method developed by Rocchio, but then reweighted all the query terms using a revised term distribution method developed earlier [YU76]. The most improvement came from reweighting, with a smaller increment from the added terms. Porter [PORTER82] implemented the CUPID retrieval system using a different reweighting scheme [ROBERTSON76] and query expansion, with user selection of the additional query terms from a set derived using relevance feedback, but did no extensive evaluation to the author's knowledge.

Smeaton[SMEATON83] tried three methods of query expansion, one based on nearest neighbors to query terms, one based on maximal spanning trees (a variation on nearest neighbors) for query terms, and one based on terms from documents retrieved using relevance feedback. He found similar performance between maximal spanning trees and nearest neighbor expansion, with worse performance using relevance feedback, and generally little improvement using any method. One problem he noted was the large number of terms added when taking all terms from the retrieved relevant, and he attempted unsuccessfully to select only the "best" terms to add, using several methods.

A modification of Porter and Smeaton's approaches seemed to be the most promising, with the user selecting terms from a small subset of the terms derived from relevance feedback. The reweighting of terms using feedback, although clearly a desirable technique, was beyond the scope of this experiment.

2.2. Methodology

The goal of this part of the experiment was to discover a method to filter the large number of additional terms provided by relevance feedback to a useful subset of about 20 terms for the interactive window. User interaction was simulated using the Cranfield 1400 test collection, with 1400 abstracts and 225 queries. Full words were used, and all queries were used in evaluation, not just those retrieving relevant documents on the first pass. The "frozen" method of evaluation was used [SALTON70], in which the top ten documents retrieved in the initial pass retain their ranks, making 11 the highest possible rank for the first feedback iteration. The methodology for feedback was as follows:

1) Run the best available ranking method for a query (see Appendix) and note which of the top ten documents retrieved for that query are relevant;

2) Create a file containing all non-common words from those relevant documents, including statistics concerning those terms;

3) Sort this term list based on a given statistical technique;

4) Add 20 terms from the top of the sorted list to the query;

5) Rerank the expanded query against the collection of unretrieved documents (no reweighting of query terms), and evaluate using the "frozen" method;

6) Repeat 3, 4, and 5 for each different statistical technique.

2.3. Results

The initial feedback results are shown in Table 1. Six different statistical techniques were used for sorting the list.

1) noise
2) postings
3) noise within postings
4) noise * frequency within postings
5) noise * frequency * postings
6) noise * frequency

The variable noise represents the given term's noise (see Appendix) based on the entire database, and the sort goes from lowest noise to highest noise. The variable postings is the number of postings within the set of relevant documents used for feedback, that is, the number of relevant documents within the top ten documents retrieved in the initial pass that contained the given term, and the sort goes from highest number of postings to lowest. Similarly the frequency is the total frequency of the given term within the set of relevant documents used for feedback, and the sort goes from highest total frequency to lowest.

TABLE 1

FEEDBACK PERFORMANCE USING CRANFIELD 225 AND ADDING TWENTY TERMS

		Variables For Sorting						
			1	2	3	4	5	6
		no feedback	noise	post	noise w/in post	noise*freq w/in post	noise*freq * post	noise*freq
A	% imp.avg. prec.	0	5.3	5.2	7.7	8.8	9.4	8.1
B	rel. by 10	650	650	650	650	650	650	650
	rel. by 20	830	905	909	945	954	962	951
	% improvement	0	41.7	43.9	63.9	68.9	73.3	67.2
	rel. by 30	946	1002	1031	1060	1076	1086	1076
	% improvement	0	18.9	28.7	38.5	43.9	47.3	43.9
C	queries imp.	0	70	82	94	94	91	92
	queries dec.	0	36	24	25	25	24	22

Three types of evaluation measures are shown in the table, (A) the average improvement in precision based on recall-level averages [SALTON83], (B) the counts of the number of relevant documents retrieved by a given document cutoff, and (C) the number of queries that show improvement or degradation in performance after thirty documents have been seen (20 previously unseen documents). As can be seen from the table, even the worst performance added 75 (905-830) more relevant documents by a cutoff of 20 documents retrieved than would have been retrieved had no terms been added, and the best performance added 132 more relevant, or an improvement of 73.3% over no feedback. This magnitude of performance improvement is not reflected in the change in average precision because of the requirement to freeze ranking for the initial retrieval. The improvement for feedback is clearly significant, with the best results showing performance improvement in 91 queries, a decrement in performance for only 24, and 110 queries with no change in performance.

The statistical technique used for sorting had a large effect on performance. Sorting the terms based strictly on their noise within the entire database (sort 1) performed the worst, indicating that the distribution of a term within the entire database is not very predictive of its value in query expansion. Using the number of postings of a term within the set of relevant documents used for feedback (sort 2) worked much better, as a term appearing in most of those relevant documents usually represents a concept central to those documents. However, the random ordering of the terms within a given number of postings is not as effective as a sort by noise within the number of postings (sort 3). The fourth and fifth sorts further refine the method by adding the effect of the frequency of the term within the set of relevant documents used for feedback. A term that appears frequently in a document is often an important term in that document, and this concept can be extended to a group of documents. In both sorts the frequency of the term within the set is used with a $\log_2$ function to dampen the effect. For example, the actual formula used for sort 5 is the noise of a term times the $\log_2$ of its frequency within the set times the number of postings within the set. Sort 6 eliminates the number of postings, and is not quite as effective, even though the frequency is related to the number of postings. As might be expected, there is significant performance improvement between the best sort, sort 5, and the worst sort, sort 1. However, although there appear to be differences in performance between the better sorts (sorts 3, 4, and 5), these are not significant based on a sign test using the number of queries that show improvements or degradations in performance.

2.4 Expanding the Number of Terms from Feedback

Whereas twenty terms seems to be an appropriate number of terms to provide for user selection, it is possible to show more than twenty terms to a user by employing a scrolling mechanism in the window. These additional terms must add significant performance improvement over the initial twenty terms, however, to justify the additional user effort. Experiments were run using the six sorting techniques and varying the number of terms added from only ten terms to forty terms. A sample of the results are shown in Figure 2, where the data from the worse sort (labeled "W") and the best sort (labeled "B") are plotted.

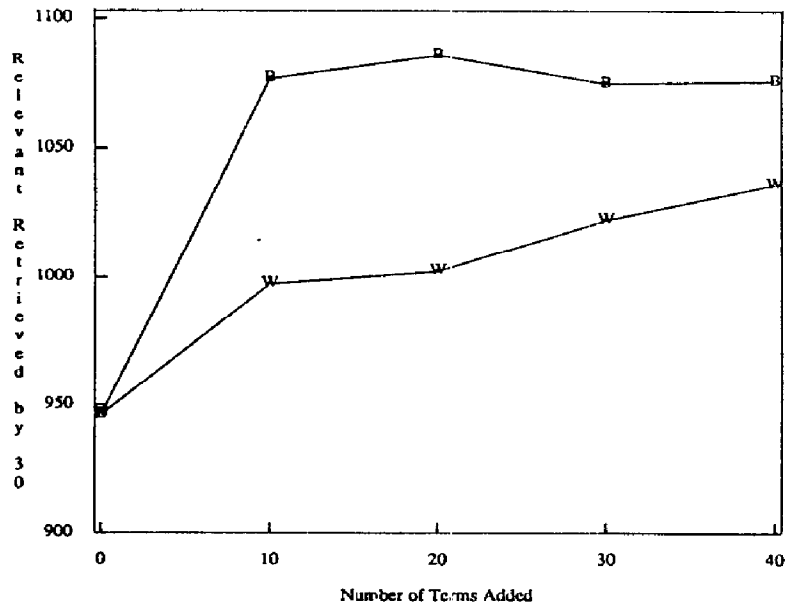

Figure 2 -- Feedback Performance as a Function of the Number of Terms Added to a Query

These results are typical for the other sorts in that the better sorts reached their peak performance after only twenty terms were added, with performance slowly decreasing after that. The poorer sorts produced poor performance initially, but continued to improve as more terms were added, although never above the performance of the better sorts. This is a reflection on the effectiveness of the sorts in that the better sorts put the terms most useful for retrieval (for a given query) at the top of the list, and adding terms beyond the top twenty tended to add terms that were less useful.

The lack of improvement from adding more than twenty terms shows that the methods developed do act as an effective filter for the large number of terms provided by relevance feedback (an average of 162 per query), and suggests that previous lack of improvement from relevance feedback expansion may have been caused by too many terms being added to the query.

2.5 User Filtering

The large improvement in performance obtained by expanding the query by the top twenty terms in the sorted list measures the improvement if all terms from the window are added. In an interactive situation, the user will select only those terms deemed useful. An effort was made to see what further improvement this could bring by adding only those terms in the window that actually appear in relevant documents that have not been seen by the user. This represents a "perfect" choice by the user. The filtering reduced the set of twenty terms added to an average of 12 terms per query. The performance results were much improved, adding 97 more relevant documents to the total relevant retrieved by a cutoff of twenty documents, an improvement of 31% over performance with no user selection. User selection would not be perfect, of course, but most users will be able to eliminate many terms that are clearly unrelated to the query.

3. The Use of Suffixing to Produce Term Variants for Query Expansion (Window 2)

3.1 Past Work

The conflation of word variants using suffixing algorithms is one of the earliest enhancements to statistical keyword retrieval systems [SALTON68]. These systems automatically expand the query by its term variants, and use term weighting based on the root of the word rather than the word itself. Three stemming algorithms, the Lovins algorithm [LOVINS68], the Porter algorithm [PORTER80], and a singular-plural conflation algorithm, have become widely used in the retrieval community. In terms of retrieval performance, the Lovins and Porter algorithms were compared [LENNON81] using several collections, including the Cranfield collection (titles only) and were found to make very little difference in retrieval performance from that using full word retrieval. Further work using the Cranfield 1400 collection abstracts [HARMAN87] revealed that the addition of term variants from stem conflation does improve performance, but only for some queries. Overall performance on available test collections showed no significant improvement from the addition of term variants. A large number of queries have lower performance because many of the added term variants appear in no relevant documents for that query, causing a drop in precision. No technique was found for separating the useful term variants (for a given query) from the non-useful ones, and it was proposed that an interactive system might allow the user to provide the necessary separation.

3.1 *Methodology*

The goal of this part of the experiment was to discover a method to separate the useful term variants from the non-useful ones, or, failing in that, to estimate how much improvement can be gained from adding term variants that have been selected by the user. The methodology was as follows:

1) Run the best available ranking method for a query using full words;

2) Add all term variants of query terms to the query using the Lovins stemming algorithm;

3) Rerank the expanded query against the collection of unretrieved documents (no reweighting of query terms), and evaluate using the "frozen" method.

3.2 Results

The term variant expansion results are shown in Table 2.

TABLE 2
QUERY EXPANSION USING TERM VARIANTS, CRANFIELD 225

	Filtering Technique				
	no expansion	no filtering	feedback filtering	user filtering	feedback and user filtering
% imp.avg. prec.	0	-2.8	1.8	8.0	4.1
rel. by 10	650	650	650	650	650
rel. by 20	830	793	856	944	892
% improvement	0	-25.8	14.4	63.3	34.4
rel. by 30	946	832	973	1075	1007
% improvement	0	-27.6	9.1	43.6	20.6
queries imp.	0	33	40	91	50
queries dec.	0	77	32	17	18

The addition of term variants using the Lovins stemmer and no filtering produced a significant decrement in performance. This is consistent with past experience using stemmers, in that many of the term variants being added are not useful for retrieval (for the given query) and only cause precision drops. The performance decrement is somewhat smaller using the Porter algorithm or the singular-plural conflation algorithm (results not shown), but these stemmers add an average of only 17 and 5 terms respectively (for the Cranfield collection), and more possible expansion terms were needed for the window (Lovins adds an average of 39 terms per query).

Various filtering attempts were tried. Past work [HARMAN87] showed that filtering using the noise of a term in a database was not effective. The first new filtering technique tried was using the terms found in retrieved relevant documents (those retrieved in the first pass) as a filter on the term variants, i.e. only term variants that appear in those relevant documents are used for expansion. This proved to be a somewhat effective filter, producing significant performance improvement over no filtering, but minimal performance improvement over no query expansion by term variants.

User filtering, simulated in a similar manner to section 2.5, shows very significant performance improvement, however. This supports the proposal that a user could provide the necessary separation between useful and non-useful term variants, and additionally indicates that proper selection of term variants can produce performance improvements comparable to those seen using feedback. Again, whereas a user will not match the perfect selection performance, a substantial improvement should be possible.

A final experiment was run (feedback and user filtering) to test if a user should be shown the full set of Lovins term variants, or the feedback filtered set (an average of 3.2 terms per query). Clearly the full set is needed, as performance decreases significantly if a user is only given the limited set to select from.

4. The Use of Nearest Neighbor Routines for Query Expansion (Window 3)

4.1 Past Work

The use of thesaurii requires the availability of a costly manually-built thesaurus, an option not normally available. Attempts have been made to construct variations on automatic thesaurii, such as term-term correlation matrices, maximal spanning trees [HARPER78,VAN RIJSBERGEN81], or nearest neighbor expansions [WILLETT81], with some success. Smeaton [SMEATON82] did multiple experiments using the maximal spanning tree and the nearest neighbor techniques on the NPL test collection. Performance using the nearest neighbor expansion method was similar to that using the maximal spanning tree method, with little improvement noted for either. Smeaton suggested that the nearest neighbor to each term be added, rather than nearest neighbors to the group of terms in the query, and that not all terms need be expanded. These ideas were built on for the current work.

4.2 Methodology

The goal of this part of the experiment was to find effective nearest neighbor techniques to provide around twenty terms to fill window 3. The nearest neighbor algorithm described in [WILLETT81] was implemented, using the Dice coefficient as the similarity measure. This algorithm computes the relatedness of a given term to all other terms in the database, based on patterns of co-occurrance. A ranked list of related terms (nearest neighbors) is created, with the degree of relatedness calculated using the Dice coefficient. The methodology was as follows:

1) Run the best available-ranking method for a query using full words;

2) Calculate the nearest neighbors for each non-common term in the query;

3) Select some of the nearest neighbors from this list and add these terms to the query;

4) Rerank the expanded query against the collection of unretrieved documents (no reweighting of query terms), and evaluate using the "frozen" method.

4.3 Results

Four different selection techniques were tried, and the results are shown in Table 3.

TABLE 3
QUERY EXPANSION USING NEAREST NEIGHBORS, CRANFIELD 225

	Selection of Nearest Neighbors				
	no expansion	top 2 thres 8	top 2 thres 6	top 1 thres 6	top 5 thres 6 filtered
% imp.avg. prec.	0	-1.7	-0.9	-0.6	4.7
rel. by 10	650	650	650	650	650
rel. by 20	830	810	825	826	905
% improvement	0	-12.5	-2.9	-2.3	41.7
rel. by 30	946	900	918	929	1016
% improvement	0	-18.4	-10.4	-6.1	23.6
queries imp.	0	41	43	32	62
queries dec.	0	66	58	47	18

The first selection method uses the top 2 nearest neighbors for each term in the query that has a noise below a threshold of 8, which includes almost all terms except those with very high noise (nearly common terms). This selection method adds an average of 15 terms per query, and shows a decrement in performance from no expansion at all. The cause of the decrement is similar to the problem found when adding term variants, i.e., too many terms are being added to the query that are not useful for retrieval. The second selection technique adds fewer terms (an average of 8 per query) because query terms having a noise above 6 are not expanded. The results are better than the first technique, with fewer queries having a degradation in performance, but still no improvement over no query expansion. The third technique continues to use the lower noise threshold, but only adds the top nearest neighbor, cutting the number of terms being added to 4 on average. This further lowers the number of queries having a degradation in performance, but also lowers the number of queries showing improvement.

Clearly more terms need to be used for expansion, but not the set of terms selected by either the first technique or the second one. The final technique used added the top 5 nearest neighbors (an average of 20 new terms per query), but filtered these using the feedback terms, i.e. only those nearest neighbor terms that appear in the relevant documents retrieved on the first pass are used for expansion. This technique worked well for term variant filtering, and also works well here, showing significant improvement in performance, with 62 queries improving to only 18 queries showing a decrement in performance, compared to no query expansion.

A further set of experiments were run to determine how much improvement could be gained by user selection. Again hindsight was used to represent the "perfect choice" (as in section 2.5), and the results are shown in Table 4.

TABLE 4					
QUERY EXPANSION USING USER FILTERED NEAREST NEIGHBORS, CRANFIELD 225					
	Selection of Nearest Neighbors				
	no expansion	u.f. top 2 thres 6	u.f. top 1 thres 6	u.f. top 5 thres 6	u.f.top 5 thres 6 filtered
% imp.avg. prec.	0	6.0	3.6	8.7	7.3
rel. by 10	650	650	650	650	650
rel. by 20	830	911	889	957	938
% improvement	0	45.0	32.8	70.6	60.0
rel. by 30	946	1019	992	1079	1050
% improvement	0	24.6	15.5	44.9	35.1
queries imp.	0	68	47	95	77
queries dec.	0	21	17	19	14

As might be expected, the performance improvement was significant, with all user filtered nearest neighbor techniques performing significantly better than no expansion. The best performance was using the top 5 nearest neighbors, without the feedback filtering applied, but with user filtering. The user filtering reduced the average of 20 added terms per query to 4, and produced performance comparable to feedback expansion.

5. Combining Windows

The sample query screen shown in Figure 1 is the result of an actual expansion of a Cranfield query (query 2) using the best feedback sorting technique, the Lovins stemmer, and the top 5 nearest neighbors for all query terms with a noise below 6. Two facts are apparent from this example. First, most users can easily eliminate many words clearly of little use in retrieval for this query, such as "respect", "new", "discussed", "highly", "appears", "certain", and "having", along with technical terms not relevant to the query, such as "tools", "vtol", "fuselage", "aeroelastician" etc. This indicates that most users should experience performance closer to the "perfect choice" results shown for user filtering than to the nonfiltered results.

The second fact is the overlap of terms between tables. For example, the terms "structure" and "structures" appear in all three tables. Whereas the overlap is not high, the total performance using all three windows will be effected. Table 5 shows the effects of adding the term variants window to the feedback window, adding the nearest neighbor window to the feedback window, and finally using all three windows. The results are using the user filtered method in all cases.

TABLE 5							
QUERY EXPANSION COMBINING VARIOUS METHODS, CRANFIELD 225							
		Expansion method, user filtered					
	no expansion	feedback	term variant	nearest neighbor	feedback + term v	feedback + nn	feedback + term v + nn
% imp.avg. prec.	0	16.0	8.0	8.7	19.4	18.5	20.8
rel. by 10	650	650	650	650	650	650	650
rel. by 20	830	1059	944	957	1088	1093	1123
% improvement	0	127	63.3	70.6	143	146	163
rel. by 30	946	1165	1075	1079	1222	1213	1249
% improvement	0	122	91	95	93	90	102
queries imp.	0	122	91	95	147	146	156
queries dec.	0	11	17	19	10	9	8

As can be seen, using two types of query expansion together produces results significantly higher than a single method. In particular, adding user filtered term variants to user filtered feedback improves 25 more queries than user filtered feedback alone, and shows a 7% improvement in performance by a cutoff of twenty documents. Adding user filtered nearest neighbors to user filtered feedback improves 24 more queries, and shows an 8% improvement in performance by the same cutoff. Using all three windows improves 34 more queries than user filtered feedback alone, and shows a 16% improvement in performance by twenty documents retrieved. The total improvement over no query expansion, as shown in Table 5, is very substantial.

It is interesting to note that whereas the improvements to performance by a cutoff of twenty are relatively additive, the other evaluation measures indicate that combining all three methods of query expansion shows less improvement than might be expected. This seems to indicate that even though the terms being added are different, the same subset of documents are being retrieved. This subset of documents are those that are closely related to the query, but not closely enough related to be retrieved in the first pass. Possibly the 1249 relevant documents retrieved by rank 30 using this multi-window expansion method are nearly all that can be moved into high ranks by this technique. Other relevant documents either have poor abstracts containing few significant words, or are related to the query by some concept not easily translated into new query terms, at least not by these methods.

6. Conclusions

It has been shown that it is possible to generate short lists of terms that, when selectively added to a query, offer retrieval performance improvements that are very substantial. In particular, techniques have been developed to automatically select twenty feedback terms (from a set averaging around 160 in Cranfield) such that expanding the query by these terms provides significant performance improvement over no query expansion. It has further been shown that user selection from term variants and nearest neighbors of query terms can provide terms for query expansion that improve performance to that comparable with feedback. All three expansion techniques combined offer the user three lists of terms that can improve performance on the second pass by more than 160% over performance with no query expansion.

The costs of these techniques, both in terms of storage and response time, can range from minimal to substantial. The term variants and nearest neighbors of all database terms can be precalculated and stored in auxiliary files of sizes related to the number of unique terms in the database, usually a reasonable amount of storage, and the response time will not be significantly effected for users. The relevance feedback techniques, however, require assembling all terms contained in each retrieved relevant document, and this information is not easily available in most large-scale retrieval systems based on inverted files. Even if this information could be efficiently retrieved, the necessary retrieval and calculation to provide the merged, ranked term list could substantially effect response time. This additional cost remains a barrier to the use of relevance feedback for large-scale retrieval systems.

The techniques presented in this paper offer interactive retrieval systems the ability to provide constructive guidance to users during query modification. Even if no feedback were possible (due to either the high cost or the fact that no relevant documents were retrieved in the first pass), the term variant and nearest neighbor windows can be very effective in turning the query into a more productive set of terms. The use of short lists should encourage user participation, and the effectiveness of the new added terms should further this participation.

Acknowledgments

Thanks to the rest of my co-workers on the IRX team for providing a system so well adapted to research. In particular thanks to Rand Huntzinger for his help with UNIX, to Dennis Benson and Charles Goldstein for their helpful comments on this paper, and to Steve Pollitt for his stimulating discussions. Also thanks to former team member Larry Fitzpatrick for the nearest neighbor routine, and to the SMART project for the use of the evaluation and stemmer code.

REFERENCES

[CROFT83] Croft W.B., "Experiments with Representation in a Document Retrieval System", Information Technology: Research and Development 2 (1983), pp. 1-21.

[DENNIS64] Dennis S.F., "The Construction of a Thesaurus Automatically from a Sample of Text",Symposium Proceedings, Statistical Association Methods for Mechanized Documentation, 1964. (National Bureau of Standards Miscellaneous Publication 269).

[HARMAN86] Harman D., "An Experimental Study of Factors Important in Document Ranking", Proceedings of the 1986 ACM Conference on Research and Developments in Information Retrieval, Pisa, 1986.

[HARMAN87] Harman D., "A Failure Analysis on the Limitation of Suffixing in an Online Environment", Proceedings of the Tenth Annual International Conference on Research and Developments in Information Retrieval, New Orleans, 1987.

[HARPER78] Harper D.J. and van Rijsbergen C.J., "An Evaluation of Feedback in Document Retrieval using Co-Occurrence Data", Journal of Documentation, Vol. 34, No. 3, pp. 189-216, 1978.

[LENNON81] Lennon M., Peirce D., Tarry B. and Willett P., "An Evaluation of Some Conflation Algorithms for Information Retrieval", Journal of Information Science 3, pp. 177-188, 1981.

[LOVINS68] Lovins J.B., "Development of a Stemming Algorithm", Mechanical Translation and Computational Linguistics 11, March 1968, pp. 22-31.

[PORTER80] Porter M.F., "An Algorithm for Suffix Stripping", Program, Vol. 14, July 1980, pp. 130-137.

[PORTER82] Porter M.F., "Implementing a Probabilistic Information Retrieval System", Information Technology: Research and Development 1 (1982), pp. 131-156.

[ROBERTSON76] Robertson S.E. and Sparck Jones K., "Relevance Weighting of search terms", Journal of the ASIS, Vol. 27, No. 3, pp. 129-146, 1976.

[ROCCHIO68] Rocchio J.J., Jr., Relevance Feedback in Information Retrieval, Chapter 14 in [SALTON68].

[SALTON68] Salton G., The SMART Retrieval System--Experiments in Automatic Document Processing, Prentice-Hall, Englewood Cliffs, N.J. 1968.

[SALTON70] Salton G., "Evaluation Problems in Interactive Information Retrieval", Information Storage Retrieval, Vol. 6, pp. 29-44, 1970.

[SALTON83] Salton G. and McGill M., Introduction to Modern Information Retrieval, McGraw-Hill Book Company, New York, 1983.

[SMEATON82] Smeaton A.F., The Retrieval Effects of Query Expansion on a Feedback Document Retrieval System, Technical Report 2, Department of Computer Science, University College Dublin (1982).

[SMEATON83] Smeaton A.F. and van Rijsbergen C.J., "The Retrieval Effects of Query Expansion on a Feedback Document Retrieval System", The Computer Journal, Vol. 26, pp. 239-246, 1983.

[SPARCK JONES72] Sparck Jones K., "A Statistical Interpretation of Term Specificity and Its Application in Retrieval", Journal of Documentation, Vol. 28, No. 1, March 1972, pp. 11-20.

[VAN RIJSBERGEN81] van Rijsbergen C.J., Harper D.J. and Porter M.F., "The Selection of Good Search Terms", Information Processing and Management, Vol. 17, pp. 77-91, 1981.

[WILLETT81] Willett P., "A Fast Procedure for the Calculation of Similarity Coefficients in Automatic Classification", Information Processing and Management, Vol. 17, pp. 53-60, 1981.

[WU81] Wu H. and Salton G., "The Estimation of Term Relevance using Relevance Feedback", Journal of Documentation, Vol. 37, No. 4, December 1981, pp. 194-214.

[YU76] Yu C.T. and Salton G., "Precision Weighting--An Effective Automatic Indexing Method", Journal of the ACM, Vol. 23, No. 1, PP. 76-88, 1976.

APPENDIX

$$\text{weight}_j = \sum_{k=1}^{Q} \frac{(\log_2 \text{Freq}_{jk} \times (\text{noise}_{max} - \text{noise}_k))}{\log_2 M}$$

where Q = the number of terms in the query
Freq_{jk} = the frequency of query term k in record j
noise_{max} = the maximum value of noise_k for a given database
M = the number of terms in the record j

$$\text{noise}_k = \sum_{i=1}^{N} \frac{\text{Freq}_{ik}}{\text{TFreq}_k} \log_2 \frac{\text{TFreq}_k}{\text{Freq}_{ik}}$$

where N = the number of records in the database
Freq_{ik} = the frequency of term k in record i
TFreq_k = the total frequency of term k in the database

Interactive Query and Search in Semistructured Databases*

Roy Goldman and Jennifer Widom

Stanford University
{royg,widom}@cs.stanford.edu
http://www-db.stanford.edu

Abstract. Semistructured graph-based databases have been proposed as well-suited stores for World-Wide Web data. Yet so far, languages for querying such data are too complex for casual Web users. Further, proposed query approaches do not take advantage of the interactive nature of typical Web sessions—users are proficient at iteratively refining their Web explorations. In this paper we propose a new model for interactively querying and searching semistructured databases. Users can begin with a simple keyword search, dynamically browse the structure of the result, and then submit further refining queries. Enabling this model exposes new requirements of a semistructured database management system that are not apparent under traditional database uses. We demonstrate the importance of efficient keyword search, structural summaries of query results, and support for inverse pointers. We also describe some preliminary solutions to these technical issues.

1 Introduction

Querying the Web has understandably gathered much attention from both research and industry. For searching the entire Web, search engines are a well-proven, successful technology [Dig97,Ink96]. Search engines assume little about the semantics of a document, which works well for the conglomeration of disparate data sources that make up the Web. But for searching within a single Web site, a search engine may be too blunt a tool. Large Web sites, with thousands of pages, are attracting millions of users. The ESPN Sports site (espn.com), for example, has over 90,000 pages [Sta96] and several million page views a day [Sta97]. As large as some sites may be, they are fundamentally different from the Web as a whole since a single site usually has a controlled point of administration. Thus, it becomes possible to consistently assign and expose the site's semantic data relationships and thereby enable more expressive searches.

Consider any of the large commercial news Web sites, such as CNN (cnn.com), ABC News (abcnews.com), etc. Currently, users have very limited querying ability over the large amounts of data at these sites. A user can browse the hard-coded menu system, examine a hand-made subject index, or use a keyword-based

* This work was supported by the Air Force Rome Laboratories and DARPA under Contracts F30602-95-C-0119 and F30602-96-1-031.

P. Atzeni, A. Mendelzon, and G. Mecca (Eds.): WebDB'98, LNCS 1590, pp. 52–62, 1999.

search engine. When looking for specific data, traversing the menus may be far too time consuming, and of course a hand-made subject index will be of limited scope. Finally, while a keyword search engine may help locate relevant data, it doesn't take advantage of the conceptual data relationships known and maintained at the site. For example, at any such Web site today there is no convenient way to find:

- All photos of Bill Clinton in 1997
- All articles about snow written during the summer
- All basketball teams that won last night by more than 10 points

Such queries become possible if most or all of the site's data is stored in a database system. Recently, researchers have proposed *semistructured* data models, databases, and languages for modeling, storing, and querying World-Wide Web data [AQM+97,BDHS96,BDS95,FFLS97,MAG+97]. Such proposals argue that a graph-based semistructured database, without the requirement of an explicit schema, is better suited than traditional database systems for storing the varied, dynamic data of the Web. So far, however, there has been little discussion of who will query such data and what typical queries will look like. Given the domain, we believe that a large and important group of clients will be casual Web users, who will want to pose interesting queries over a site's data.

How would a typical Web user pose such queries? Asking casual users to type a query in any database language is unrealistic. It is possible to handle certain queries by having users fill in hard-coded forms, but this approach by nature limits query flexibility. Our previous work on *DataGuides* [GW97] has proposed an interactive query tool that presents a dynamic structural summary of semistructured data and allows users to specify queries "by example." (*Pesto* [CHMW96] and *QBE* [Zlo77], designed for object-relational and relational databases, respectively, enable users to specify queries in a similar manner.) A DataGuide summarizes all paths through a database, starting from its root. While such dynamic summaries are an important basic technology for several reasons [GW97], presenting the user with a complete summary of paths may still force him to explore much unnecessary database structure.

In this paper, targeting casual users, our strategy is to model and exploit two key techniques that Web users are intimately familiar with:

1. specifying a simple query to begin a search, usually with keywords
2. further exploring and refining the results

For the first technique, we want to support very simple queries that help "focus" the user on relevant data. The many search engines on the Web have shown that keyword search is an easy and effective technique for beginning a search. To enable the second technique, we want to expose and summarize the structure of the database "surrounding" any query result. To do this, we dynamically build and present a DataGuide that summarizes paths not from the database root, but instead from the objects returned in the query result. A user can then repeat

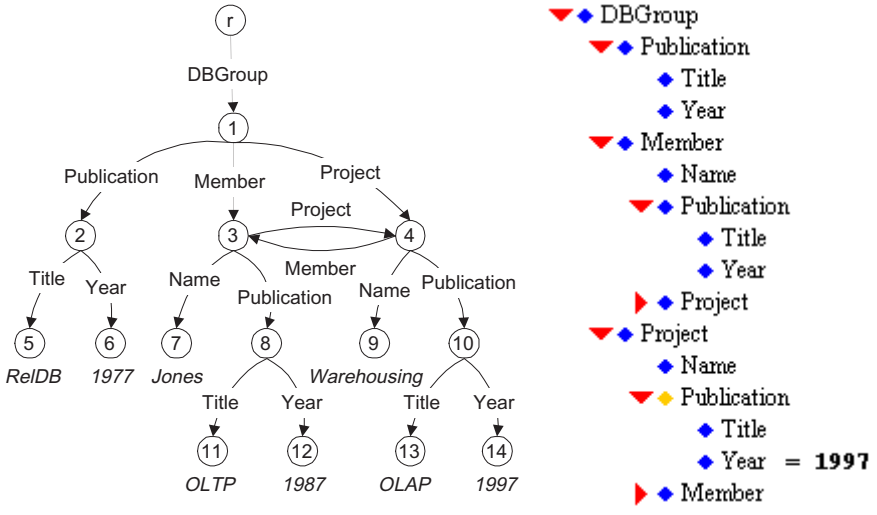

Fig. 1. A sample OEM database and its Java DataGuide

the process by submitting a query from this "focused" DataGuide or specifying additional keywords, ultimately locating the desired results.

Our discussions are in the context of the *Lore* project [MAG+97], which uses the *OEM* graph-based data model [PGMW95] and the *Lorel* query language [AQM+97]. Our results are applicable to other similar graph-based data models, as well as the emerging XML standard for defining the semantic structure of Web documents [Con97].

In the rest of the paper, we first provide background and context in Section 2. In Section 3, we present a simple motivating example to illustrate why new functionality is needed in a semistructured database system to support interactive query and search. Our session model is described in Section 4, followed by three sections covering the new required technology:

- Keyword search (Section 5): Efficient data structures and indexing techniques are needed for quickly finding objects that match keyword search criteria. While we may borrow heavily from well-proven information retrieval (IR) technology, the new context of a graph database is sufficiently different from a simple set of documents to warrant investigation.
- DataGuide enhancements (Section 6): Computing a DataGuide over each query result can be very expensive, so we have developed new algorithms for computing and presenting DataGuides piecewise, computing more on demand.
- Inverse pointers (Section 7): To fully expose the structural context of a query result, it is crucial to exploit inverse pointers when creating the DataGuide for the result, browsing the data, and submitting refining queries. While support for inverse pointers may seem straightforward, the major propo-

sed models for semistructured data are based on directed graphs, and inverse pointers have not been considered in the proposed query languages [AQM+97,BDHS96,FFLS97].

2 Background

To set the stage for the rest of the paper, we briefly describe the OEM data model, introduce the Lorel query language, and summarize DataGuides. In OEM, each object contains an object identifier (oid) and a value. A value may be atomic or complex. Atomic values may be integers, reals, strings, images, or any other indivisible data. A complex OEM value is a collection of OEM subobjects, each linked to the parent via a descriptive textual label. An OEM database can be thought of as a rooted, directed graph. The left side of Figure 1 is a tiny fictional portion of an OEM database describing a research group, rooted at object r.

The Lorel query language, derived from OQL [Cat94], evaluates queries based on *path expressions* describing traversals through the database. Special edges coming from the root are designated as *names*, which serve as entry points into the database. In Figure 1, DBGroup is the only name. As a very simple example, the Lorel query "Select DBGroup.Member.Publication.Title" returns a set containing object 11, with value "OLTP." More specifically, when a query returns a result, a new named Answer object is created in the database, and all objects in the result are made children of the Answer.[1] The Answer edge is available as a name for successive queries.

A DataGuide is a dynamic structural summary of an OEM database. It is an OEM object G that summarizes the OEM database (object) D, such that every distinct label path from the root of D appears exactly once as a path from the root of G. Further, every path from the root of G corresponds to a path that exists in D. We have carefully chosen Figure 1 to be a DataGuide of itself (ignoring atomic values). For any given sequence of labels, there is only one corresponding path in the database. (In a real database, there may be many Member objects under DBGroup, several Publication objects per Project, etc.) Through a Web-accessible Java interface, a DataGuide is presented as a hierarchical structure, and a user can interactively explore it. The right side of Figure 1 shows the Java DataGuide for our sample database. Clicking on an arrow expands or collapses complex objects. We have expanded most of the links, but because of the cycle we have not expanded the deepest Project or Member arrows.

Users can also specify queries directly from the Java DataGuide with two simple steps: 1) selecting paths for the query result, and 2) adding filtering conditions. Each diamond in the DataGuide corresponds to a label path through the database. By clicking on a diamond, a user can specify a condition for the

[1] Identifying labels are assigned to the edges connecting the Answer object to each query result object, based on the last edge traversed to reach the result object during query evaluation. In this example, the label is Title. Also, Lorel queries may create more complicated object structures as query results, but for simplicity we do not consider such queries in this paper; our work can easily be generalized.

path or select the path for the query result. Filtering conditions are rendered next to the label, and the diamonds for selected paths are highlighted. The Java DataGuide in Figure 1 shows the query to select all project publications from 1997. The DataGuide generates Lorel queries, which are sent to the Lore server to be evaluated. In our Web user interface, we format the query results hierarchically in HTML for easy browsing.

3 Motivating Example

In this section we trace a motivating example, using the sample database presented in Figure 1. Suppose a user wishes to find all publications from 1997, a seemingly simple query. (In the previous section, our sample query only found publications of projects.) It is possible to write a Lorel query to find this result, but a casual user will not want to enter a textual Lorel query. This example also illustrates some limitations of using the DataGuide to locate information. Even in this simple case, there are numerous paths to all of the publications; in a larger database the situation may be much worse. In short, while the DataGuide does a good job of summarizing paths from the root, a user may be interested in certain data independent of the particular topology of a database.

In this situation, a typical Web user would be comfortable entering keywords: "Publication," "1997," or both. Suppose for now the user types "Publication" to get started. (We will address the case where the user types "1997" momentarily, and we discuss the issue of multiple keywords in Section 5.) If the system generates a collection of all Publication objects, the answer is {2, 8, 10}, identified by the name Answer. While this initial result has helped focus our search, we really only wanted the Publications in 1997. One approach would be to browse all of the objects in the result, but again in a larger database this may be difficult. Rather, we dynamically generate a DataGuide over the answer, as shown in Figure 2. Notice now that even though Title and Year objects were reachable along numerous paths in the original DataGuide, they are consolidated in Figure 2. As shown in the Java DataGuide, the user can mark Publication for selection and enter a filtering condition for Year to retrieve all 1997 publications. Getting the same result in the original DataGuide would have required three selection/filtering condition pairs, one for each possible path to a Publication.

The above scenario motivates the need for efficient keyword search and efficient DataGuide creation over query results. Next, we show how these features essentially force a system to support inverse pointers as well. Suppose the user had typed "1997" rather than "Publication." This time, the answer in our sample database is the singleton set {14}, and the DataGuide over the result is empty since the result is just an atomic object. This example illustrates that what the user needs to see in general is the area "surrounding" the result objects, not just their subobject structure as encapsulated by the DataGuide. Given a set of objects, we can consider inverse pointers to present the "surrounding area" to the user; for example, we can give context to a specific year object by showing that it

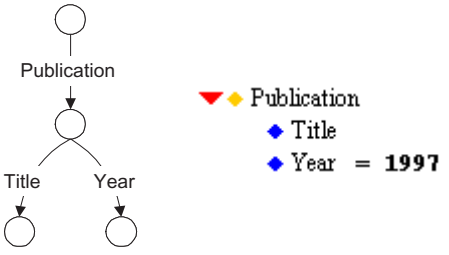

Fig. 2. DataGuide constructed over result of finding all publications

is the child of a publication object. By exploring both child and parent pointers of objects in a query result, we can create a more descriptive DataGuide.

4 Query and Search Session Model

We define a Lore *session* over an initial database D_0, with root r and initial DataGuide $G_0(r)$, as a sequence of *queries* q_1, q_2, ... q_n. A query can be a "by example" DataGuide query, a list of keywords, or, for advanced users, an arbitrary Lorel query. The objects returned by each query q_i are accessible via a complex object a_i with name $Answer_i$. After each query, we generate and present a DataGuide $G_i(a_i)$ over the result, and users can also browse the objects in each query result. Perhaps counterintuitive to the notion of narrowing a search, we do not restrict the database after each query. In fact, the database D will grow monotonically after each query q_i. After q_i, $D_i = D_{i-1} \cup a_i$. Essentially, each DataGuide helps focus the user's next query without restricting the available data. In the following three sections, we discuss three technologies that enable efficient realization of this model of interaction.

5 Keyword Search

In the IR arena, a keyword search typically returns a ranked list of documents containing the specified keywords. In a semistructured database, pertinent information is found both in atomic values and in labels on edges. Thus, it makes sense to identify both atomic objects matching the specified word(s) and objects with matching incoming labels. For example, if a user enters "Publication," we would like to return all objects pointed to by a "Publication" edge, along with all atomic objects with the word "Publication" in their data. This approach is similar in spirit to the way keyword searches are handled by Yahoo! (yahoo.com). There, search results contain both the *category* and *site* matches for the specified keywords.

While a keyword search over values and labels is expressible as a query in Lorel (and also in *UnQl* [BDHS96]), the issue of how to efficiently execute this particular type of query has not been addressed. In Lore, we have built two

inverted-list indexes to handle this type of query. The first index maps words to atomic objects containing those words, with some limited IR capabilities such as *and*, *or*, *near*, etc. The second index maps words to edges with matching labels. Our keyword search indexes currently range over the entire database, though query results can be filtered using Lorel.

An interesting issue is how to handle multiple keywords. It is limiting to restrict our searches to finding multiple keywords within a single OEM object or label, since our model encourages decomposition into many small objects. Hence, we would like to efficiently identify objects and/or edges that contain the specified keywords and are also near each other in terms of link distance. Further, we must decide how to group or rank the results of a keyword search, an essential aspect of any search engine that may return large answer sets. These issues are discussed in detail in [GSVGM98]: we formalize the notion of link distance and ranking, and we introduce specialized indexes to speed up distance computations.

6 DataGuide Enhancements

As described in the motivating example, we wish to build DataGuides over query results. For this section, let us ignore the issue of inverse pointers. As shown in [GW97], computing a DataGuide can be expensive: the worst case running time is exponential in the size of the database, and for a large database even linear running time would be too slow for an interactive session. We thus introduce two techniques to improve the running time.

First, we can exploit certain auxiliary data structures that are built to provide incremental DataGuide maintenance [GW97]. These structures guarantee that we never need to recompute a "sub-DataGuide" that has previously been constructed. In Figure 1, suppose a user searches for all "Projects," a query that would return the singleton set {4}. In this case, the DataGuide over {4} is the same as the sub-DataGuide reachable along DBGroup.Project in the original DataGuide. We can dynamically determine this fact with a single hash table lookup, and no additional computation is needed.

Second, we observe that an interactive user will rarely need to explore the entire DataGuide. Our experience shows that even in the initial DataGuide, users rarely explore more than a few levels. Most likely, after a reasonable "focusing" query, users will want to browse the structure of objects near the objects in the query result. Hence, we have modified the original depth-first DataGuide construction algorithm to instead work breadth-first, and we have changed the algorithm to build the DataGuide "lazily," i.e., a piece at a time. From the user's perspective, the difference is transparent except with respect to speed. When a user clicks on an arrow for a region that hasn't yet been computed, behind the scenes we send a request to Lore to generate and return more of the DataGuide. Our maintenance structures make it easy to interrupt DataGuide computation and continue later with no redundant work.

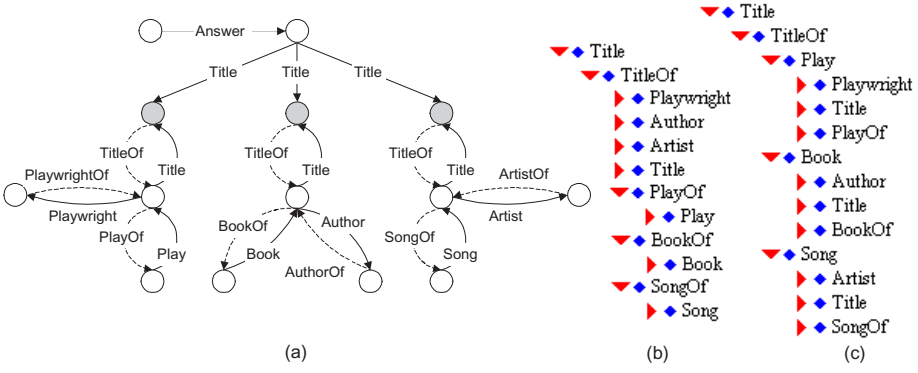

Fig. 3. An OEM query result and two potential DataGuides

7 Inverse Pointers

Directed graphs are a popular choice for modeling semistructured data, and the proposed query languages are closely tied to this model. While powerful regular expressions have been proposed for traversing forward links, essentially no language support has been given to the problem of directly traversing inverse pointers. As our motivating example demonstrates, a parent may be just as important as a child for locating relevant data.

Adding inverse pointers affects many levels of a semistructured database system, including object storage, creation of DataGuides (or any other summarizing technique), query language design, and query execution. Physically, inverse pointers may be clustered with each object's data or stored in a separate index. Logically, we try to make access to inverse pointers as seamless as possible: for an object O with an incoming label "X" from another object P, we conceptually make P a child of O via the label "XOf." With this approach, inverse edges can be treated for the most part as additional forward edges. Next, we focus on how exposing inverse pointers affects DataGuides, graph query languages, and query execution. (Note that even if inverse pointers are not added to the model or query language, they can still be very useful for "bottom-up" query processing, as described in [MW97].)

DataGuide Creation We wish to extend DataGuides to summarize a database in all directions, rather than only by following forward links. If the "Of" links described above are simply added to the database graph, then we need not even modify our DataGuide algorithms. Unfortunately, this approach can yield some unintuitive results. In OEM and most graph-based database models, objects are identified by their incoming labels. A "Publication," for example, is an object with an incoming Publication edge. This basic assumption is used by the DataGuide, which summarizes a database by grouping together objects with identical incoming labels. An "Of" link, however, does a poor job of identifying

an object. For example, given an object O with an incoming TitleOf link, we have no way of knowing whether O is a publication, book, play, or song. Therefore, a DataGuide may group unrelated objects together. For example, suppose a user's initial search over a library database finds some Title objects. Figure 3(a) shows three atomic objects in the result (shaded in the figure), with dashed "Of" links to show their surrounding structure. Figure 3(b) shows the standard DataGuide over this Answer. The problems with 3(b) should be clear: the labels shown under TitleOf are confusing, since the algorithm has grouped unrelated objects together. Further, the labels directly under TitleOf do not clearly indicate that our result includes titles of books, plays, and songs. To address the problem, we have modified the DataGuide algorithm slightly to further decompose all objects reachable along an "Of" link based on the non-"Of" links to those objects. Figure 3(c) shows the more intuitive result, which we refer to as a *Panoramic DataGuide*. Of course, since OEM databases can have arbitrary labels and topologies, we have no guarantees that a Panoramic DataGuide will be the ideal summary; still, in practice it seems appropriate for many OEM databases. Note that adding inverse pointers to DataGuide creation adds many more edges and objects than in the original DataGuide, making our new support for "lazy" DataGuides (Seciton 6) even more important.

Query Language & Execution Just as users can specify queries "by example" with the original DataGuide, we would like to allow users to specify queries with Panoramic DataGuides as well. Suppose in Figure 3(c) a user selects Author to find the authors of all books having titles in the initial result. In Lorel, which currently does not support direct access to inverse pointers, the generated query is:

```
Select A
From Answer.Title T1, #.Book B, B.Title T2, B.Author A
Where T1 = T2
```

This query essentially performs a join between the titles in our answer and all book titles in the entire database, returning the authors of each such book. The # is a "wildcard" representing any path, and because of this wildcard a naive execution strategy could be very expensive. Efficient execution based on forward pointers alone depends on having an index that quickly returns all Book objects in the database, and we do support such an index in Lore. If we store inverse pointers in the system, we might be able to train the optimizer to exploit them for such queries [MW97]; rather than finding all Book objects and performing the join, the system could simply follow inverse and then forward pointers from each Title in the initial result. However, it could be difficult to recognize and optimize these cases. Another approach is to allow inverse links to be specified directly in path expressions in the language.

As an alternative to storing inverse pointers, the query processor could "remember" the (forward) path traversed to evaluate a query. The user could then explore this path to see some of the result's context. Lore can in fact provide such

a *matched path* for each query result. However, when an execution strategy does not involve navigating paths from the root, generating a matched path from the root would drastically increase query execution time. Further, a matched path still does not allow a user to arbitrarily explore the database after a query result.

8 Implementation Status

Our interactive query and search model, along with the necessary supporting features discussed in this paper, are under development within the Lore project. We keep our online Lore demo up-to-date, reflecting new designs as they are completed. Please visit www-db.stanford.edu/lore.

References

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Canada, June 1996.

[BDS95] P. Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proceedings of the 1995 International Workshop on Database Programming Languages (DBPL)*, 1995.

[Cat94] R.G.G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Francisco, California, 1994.

[CHMW96] M. Carey, L. Haas, V. Maganty, and J. Williams. Pesto: An integrated query/browser for object databases. In *Proceedings of the Twenty-Second International Conference on Very Large Data Bases*, pages 203–214, Bombay, India, August 1996.

[Con97] World Wide Web Consortium. Extensible markup language (XML). http://www.w3.org/TR/WD-xml-lang-970331.html, December 1997. Proposed recommendation.

[Dig97] Digital Equipment Corp. About AltaVista: our technology. http://altavista.digital.com/av/content/ about_our_technology.htm, 1997.

[FFLS97] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a Web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[GSVGM98] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases*, New York, New York, 1998.

[GW97] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pages 436–445, Athens, Greece, August 1997.

[Ink96] Inktomi Corp. The technology behind HotBot. http://www.inktomi.com/Tech/CoupClustWhitePap.html, 1996. White paper.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[MW97] J. McHugh and J. Widom. Query optimization for semistructured data. Technical report, Database Group, Stanford University, November 1997. Available at URL http://www-db.stanford.edu/pub/papers/qo.ps.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[Sta96] Starwave Corp. About ESPN. http://www.starwave.com/starwave/ about.espn.html, 1996.

[Sta97] Starwave Corp. ESPN SportsZone surpasses 2 billion page views. http://www.starwave.com/ starwave/releases.sz.billion.html, December 1997. Press release.

[Zlo77] M. Zloof. Query by example. *IBM Systems Journal*, 16(4):324–343, 1977.

3. Nachtrag zum Mietvertrag vom 18./23.03.2012

zwischen

Dr. Andreas Sturm, vertreten durch Herrn Erwin Sturm, Hochstraße 18, 94032 Passau

- Vermieter -

und

Mathias Möller, Weinleitenweg 15, 94036 Passau

- Mieter -

- Der Vermieter, Dr. Andreas Sturm gestattet es den Mietern/dem Mieter die Wohnung nach voriger Absprache unter zu vermieten.

Passau, den

_____ _____

Erwin Sturm für Dr. Andreas Sturm Mathias Möller